Projet Startix

Objectif

Startix a pour objectif de faciliter et d'accélérer: * l'intégration des applications – nouvelles ou existantes – dans l'écosystème technique de notre entreprise. Il répond à un besoin récurrent : la difficulté à accéder efficacement aux offres de services internes (authentification, supervision applicative, bases de données, etc.), souvent en raison de documentations incomplètes ou trop génériques. * la contribution, par toute personne intéressée au sein de la DSIT, sur des sujets récurrents à valeur ajoutée pour les projets, n'entrant pas forcément dans le cadre d'une offre de service DSIT. Ce 2ème point peut éventuellement préparer le terrain à de nouvelles offres de service dans une approche Bottom-Up

Le projet s'adresse à plusieurs profils : - Développeurs d'applications legacy - Développeurs de nouvelles applications - Chefs de projet

Principaux livrables

Startix repose sur quatre axes concrets :

1. **Documentation technique enrichie**
    - Description détaillée des étapes internes à suivre pour consommer chaque offre de service
    - Clarification et complément de la documentation existante des services internes
2. **Tutoriels simples et pratiques**
    - Exemples concrets d'intégration dans le code (ajout de dépendances, configuration, appels d'API, etc.)
    - Focus sur des cas minimaux, reproductibles et facilement adaptables
3. **Application blanche (boilerplate)**
    - Application illustrative intégrant toutes les offres de service
    - Sert de support aux tutoriels et à la validation des intégrations
    - ⚠️ **Attention : cette application n'impose aucune architecture aux projets**. Elle est uniquement conçue comme **un exemple** pour faciliter la compréhension et l'intégration. Chaque projet reste totalement libre de son organisation technique.
4. **Écosystème collaboratif**
    - Mise en place d'un cadre permettant la contribution communautaire
    - Capitalisation des retours d'expérience et des bonnes pratiques
    - Enrichissement continu des ressources pour gagner en autonomie et en efficacité

Contributing

L'esprit de Startix est de créer et maintenir une dynamique de contribution communautaire. Dans cette perspective, les contributions ne sont pas obligées de respecter dès le début tous les livrables ci-dessus pour être acceptées. En contrepartie, elles ont pour cible de les atteindre à l'avenir pour être au même niveau d'exigence que les contributions du projet Startix. Ainsi un tableau de suivi des contributions existe ici et a pour but de favoriser l'avancement par jalon de maturité.

Structure du projet

Startix est composé d'un **frontend Angular**, d'un **backend Spring Boot**, d'un ensemble de fichiers de **déploiement Kubernetes**, et d'une **documentation technique et fonctionnelle**.

Arborescence du projet

```
├── 📁 backend / → Application backend Spring Boot
│   └── README.md → Guide technique pour lancer le backend
│
├── 📁 frontend / → Application frontend Angular
│   └── README.md → Guide technique pour lancer le frontend
│
├── 📁 deploiement / → Les Composants et les fichiers de déploiement Kubernetes et Minikube (YAML)
│   └── 📁 k8s
│
├── 📁 documentation / → Documentation fonctionnelle et technique
│   ├── 📁 IC
│   ├── 📁 DC
│   ├── 📁 INPUT
│   └── 📁 Authentification
│
└── README.md → Présentation globale du projet
```

`backend/` – Spring Boot

Contient les API REST sécurisées avec OAuth2/OIDC GAIA, avec une gestion des rôles, la logique métier, et l'intégration des services RTE.

Voir backend/README.md pour : - Installation des dépendances - Configuration de la base de données - Lancement local avec Maven ou Docker - Variables d'environnement nécessaires - Endpoints exposés

`frontend/` – Angular

Contient l'interface utilisateur Angular avec authentification OIDC GAIA, les appels API sécurisés vers le backend, et la gestion des rôles.

Voir frontend/README.md pour : - Installation des dépendances - Lancement local avec npm et docker - Lancement des tests et l'analyse Sonar - Endpoint exposé

`deploiement/` – Kubernetes

Contient les fichiers YAML pour déployer l'application sur un cluster Kubernetes et en local avec `minikube`: - Les composants de déploiement backend et frontend - Les composants de déploiement en local en utilisant `minikube` - Services, ingress, ConfigMaps et secrets (si nécessaires) pour chaque composant. - Un dossier kustomize pour chaque environnement - dev: ns-startix-dev - int: ns-startix-int - local: ns-startix-local - prod: ns-startix-prod

`documentation/` – Documentation technique et fonctionnelle

Contient les guides d'intégration des services RTE avec les demandes administratives COSMOS nécessaires.

En résumé

Startix n'est pas un framework, ni une architecture imposée. C'est une **boîte à outils pédagogique** conçue pour accompagner les équipes dans l'intégration des services internes. Il vise à réduire la courbe d'apprentissage, fluidifier les démarrages projets et encourager le partage de connaissances.

Pour toute contribution ou question, merci de consulter la section CONTRIBUTING.md ou d'ouvrir une issue.

Documentation Authentification G@IA - Ping Federate

À propos

Ce document explique comment configurer l'authentification directe avec G@IA OpenID Connect (Ping Federate). L'objectif est de permettre à l'application de déléguer l'authentification directement à G@IA OpenID Connect (Ping Federate).

Context

La sécurisation de l'authentification des utilisateurs est primordiale chez RTE. Gaïa est le serveur responsable d'authentification des internes RTE.

Gaia est basé sur la solution PingFederate, un serveur de fédération d'identité développé par Ping Identity, conçu pour gérer l'authentification, le Single Sign-On (SSO), et la fédération d'identité dans les environnements d'entreprise.

Dans le cadre du projet Startix, on a choisi le protocole **OAuth2** et **OpenID Connect** avec le flux **Authorization Code + PKCE**. C'est la combinaison la plus sécurisée, standardisée et adaptée aux applications modernes SPA + microservices.

Architecture concernée :

- Frontend (SPA) : application côté client, sans backend propre.
- Backend (micro-services) : expose des APIs REST sécurisées.
- Authentification SSO : via un fournisseur d'identité (ex. PingFederate, Keycloak, Azure AD).
- Token JWT : utilisé pour authentifier les appels API.

Pourquoi OAuth2 + OpenID Connect ?

- OAuth2 gère l'autorisation : il permet au frontend d'obtenir un access token pour appeler les APIs.
- OpenID Connect (OIDC) ajoute l'authentification : il fournit un ID token contenant l'identité de l'utilisateur (nom, email, etc.).

Ensemble, ils permettent une authentification sécurisée + accès aux ressources, avec SSO.

Pourquoi le flux Authorization Code + PKCE ?

Le flux Authorization Code permet à l'utilisateur de s'authentifier, puis un code d'autorisation est échangé contre un jeton d'accès.

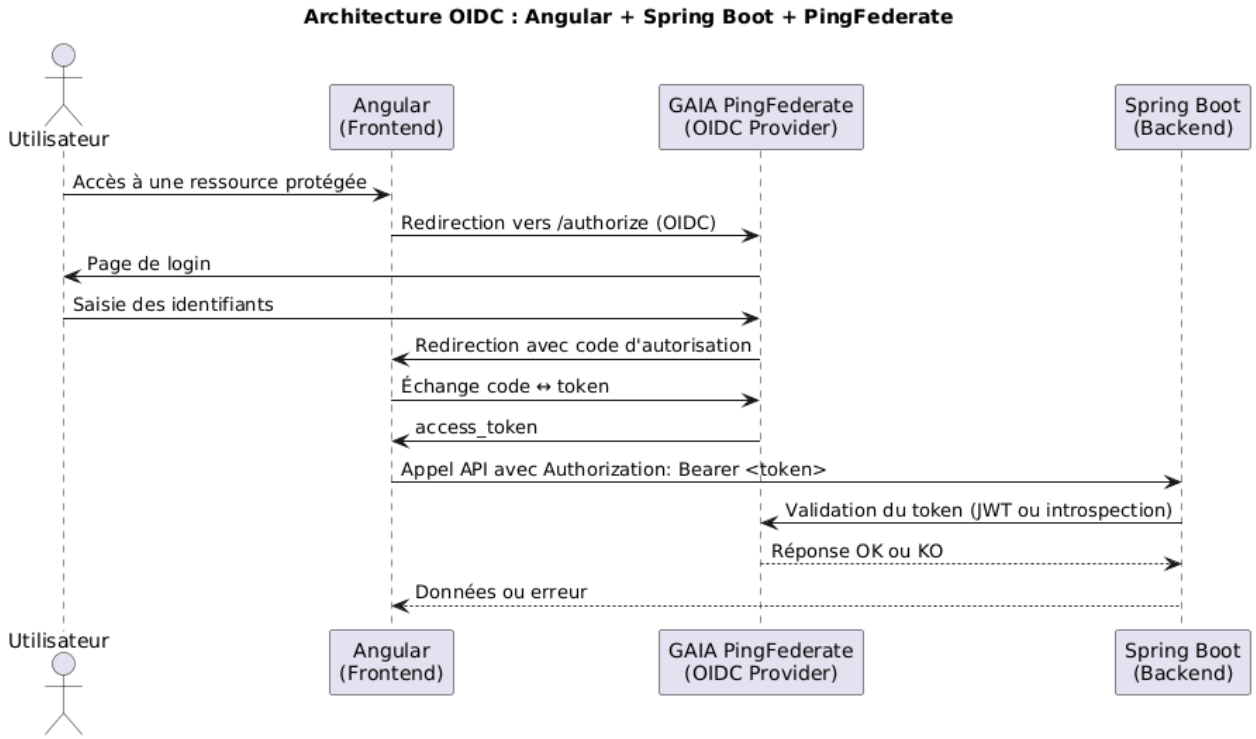
L'Authorization Code avec PKCE est la variante sécurisée du flux précédent, sans besoin de secret client.

Sécurité PKCE - Adapté aux clients publics comme Angular, qui ne peuvent pas stocker un secret. - PKCE protège contre les attaques d'interception du code d'autorisation

Validation côté G@IA - G@IA vérifie que `SHA256(code_verifier) == code_challenge` - Seule l'application qui a généré le challenge peut échanger le code

Tutoriel / Intégration

Architecture



Composants - **Frontend Angular** - Utilisation de la librairie `angular-oauth2-oidc` comme client OpenID Connect standard. - Implémente le flux Authorization Code avec PKCE. - Envoie le token dans l'en-tête Authorization des requêtes HTTP. - **Gaia PingFederate (OIDC Provider)** - Gère l'authentification des utilisateurs. - Gère les claims et les rôles. - **Backend Spring Boot** - Utilise `spring-boot-starter-security` et `spring-boot-starter-oauth2-resource-server` - Vérifie le token - Protège les endpoints REST avec des règles de sécurité et la gestion des rôles.

Flux d'Authentification OAuth2 + PKCE

🔄 **Séquence détaillée**

1. **Accès initial** : L'utilisateur accède à l'application Angular
2. **Détection** : L'application détecte que l'utilisateur n'est pas authentifié
3. **PKCE Generation** : L'application génère :
    - `code_verifier` (chaîne aléatoire)
    - `code_challenge` (SHA256 du verifier, encodé en base64url)
4. **Redirection autorisation** : L'application redirige vers G@IA avec :

```
https://gaia-sso.opf.rte-france.com/auth/oauth2/authorize?
  client_id=startix-dev-front-opf&
  redirect_uri=http://localhost:9000&
  response_type=code&
  scope=openid%20profile%20email&
  code_challenge=xyz123&
  code_challenge_method=S256&
  state=random_state
```

1. **Authentification G@IA** : L'utilisateur s'authentifie via l'interface G@IA
2. **Callback** : G@IA redirige vers l'application avec un `code` d'autorisation
3. **Échange de code** : L'application échange le code contre les tokens :

```
POST https://gaia-sso.opf.rte-france.com/auth/oauth2/token
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code&
client_id=startix-dev-front-opf&
code=AUTH_CODE&
redirect_uri=http://localhost:9000&
code_verifier=original_verifier
```

1. **Réception tokens** : G@IA retourne :
    - `access_token` (avec les rôles dans les claims)
    - `id_token` (informations utilisateur)
    - `refresh_token`
2. **Stockage du token** : L'application Angular stocke le access_token en mémoire ou session.
3. **Appels HTTP sécurisés** : Pour chaque requête vers le backend Spring Boot, Angular ajoute le token dans l'en-tête :

```
Authorization: Bearer <access_token>
```

1. **Réception de la requête** : Le backend Spring Boot reçoit la requête HTTP avec le JWT dans l'en-tête Authorization.
2. **Validation du token** : Spring Boot vérifie :
3. La signature du JWT (via JWKS du serveur OIDC)
4. L'expiration du token
5. Le format et les claims
6. **Extraction des rôles** : Spring extrait les rôles depuis le claim `roles` du JWT
7. **Contrôle d'accès** : avec @PreAuthorize Les endpoints sont protégés selon les rôles
8. **Réponse sécurisée** : Si le token est valide et les rôles autorisés, le backend retourne les données. Sinon, il renvoie :
9. 401 Unauthorized si le token est invalide
10. 403 Forbidden si l'utilisateur n'a pas les droits

Intégration Oauth2 dans Angular (Frontend)

1. Installation des dépendances

Ajouter la dépendance `"angular-oauth2-oidc": "20.0.2",` dans le fichier package.json.

2. Configuration

Ajouter les variables d'environnement dans les fichiers de configuration: - environment.local.ts: Le fichier d'environnement local (utilisé dans le code Angular). - environment.ts: Le fichier d'environnement de production (utilisé dans le code Angular). - env.js: Ce fichier est utilisé au niveau du index.html. Il permet de charger les variables d'environnement dans la variable globale du navigateur "window". - env.js.template: Ce fichier est utilisé au niveau du Dockerfile. Il prend en charge les variable d'environnement définies dans la VM. - env.conf: Le fichier de conf principal du frontend.

Exemple environment.local.ts (Développement)

```
export const environment = {
  oidc: {
    issuer: nativeWindowEnv?.OIDC_ISSUER || 'https://gaia-sso.opf.rte-france.com',
    clientId: nativeWindowEnv?.OIDC_CLIENT_ID || 'startix-dev-front-opf',
    redirectUri: nativeWindowEnv?.OIDC_REDIRECT_URI || 'http://localhost:9000/',
    scope: 'openid profile email',
  }
};
```

3. Authentification OIDC dans Angular

Le dossier frontend/src/app/auth/* contient le service Oauth2AuthService ainsi que l'intercepteur HTTP httpAuthInterceptor, utilisés pour la gestion de l'authentification OAuth2/OIDC dans l'application.

3.1 Le Service `Oauth2AuthService`

Le service `Oauth2AuthService` encapsule la logique d'authentification avec OpenID Connect via la bibliothèque `angular-oauth2-oidc`.

Configuration

- Initialise le client OAuth2 avec les paramètres du serveur OIDC (issuer, clientId, redirectUri…).
- Active le rafraîchissement silencieux du token.

Fonctions principales

Méthode	Description
`initAuthentication()`	Initialise l'authentification à partir du document de découverte OIDC.
`login()`	Lance le flux de connexion via redirection.
`logout()`	Déconnecte l'utilisateur et nettoie le token local.
`silentRefresh()`	Tente de rafraîchir le token en arrière-plan.
`token`	Retourne le token d'accès actuel.
`isAuthenticated`	Vérifie si le token est valide.
`roles`	Extrait les rôles depuis les claims OIDC.
`isAdmin`	Vérifie si l'utilisateur possède le rôle `ROLE_ADMIN`.
`userInfo`	Retourne les informations utilisateur (claims).

3.2 Intercepteur HTTP `httpAuthInterceptor`

Cet intercepteur ajoute automatiquement le token d'accès dans les requêtes HTTP vers les routes protégées.

Fonctionnement

- Vérifie si l'URL cible fait partie des routes protégées (`/bonjour`, `/api/notes`).
- Si oui, ajoute l'en-tête `Authorization: Bearer <token>` à la requête.
- Sinon, laisse passer la requête sans modification.

Exemple d'en-tête ajouté

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

4. Gestion des rôles

Claims dans l'Access Token

Avec la configuration demandée dans le ticket Cosmos, l'access token contiendra :

```
{
  "sub": "user123",
  "iss": "https://gaia-sso.opf.rte-france.com",
  "aud": "startix-dev-front-opf",
  "roles": ["ROLE_USER", "ROLE_ADMIN"],
  "email": "user@rte-france.com",
  "given_name": "Jean",
  "family_name": "Dupont"
}
```

Utilisation dans Angular

```
// Dans votre service
get roles(): string[] {
  const claims = this.oauthService.getIdentityClaims();
  return claims?.['roles'] || [];
}

get isAdmin(): boolean {
  return this.roles.includes('ROLE_ADMIN');
}

get isUser(): boolean {
  return this.roles.includes('ROLE_USER');
}
```

Intégration Oauth2 dans SpringBoot (Backend)

1. Installation des dépendances

Dans le pom.xml, ajouter les dépendances :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

2. Configuration

- Ajouter dans le fichier de configuration application.yml

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://gaia-sso.opf.rte-france.com
          jwk-set-uri: https://gaia-sso.opf.rte-france.com/pf/JWKS
```

Spring récupère automatiquement les clés publiques via .well-known/openid-configuration.

- Dans le fichier env.conf

```
## GAIA OUATH2 OIDC ##
SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_ISSUER_URI=https://gaia-sso.opf.rte-france.com
SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_JWK_SET_URI=${SPRING_SECURITY_OAUTH2_RESOURCESERVER_JWT_ISSUER_URI}/
```

3. Activer les annotations de sécurité

Dans la classe de configuration SecurityConfiguration.java

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(securedEnabled = true)
public class SecurityConfiguration {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeHttpRequests(authorize -> authorize
            .requestMatchers("/").permitAll()
            .requestMatchers("/api/notes/**").authenticated()
            .requestMatchers("/bonjour").authenticated()
            .anyRequest().authenticated()
            )
            .oauth2ResourceServer(oauth2 -> oauth2
              .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtAuthenticationConverter())) // permet de personnal
            );
        return http.build();
    }
}
```

4. Personnaliser l'extraction des rôles

Les rôles sont cofigurés dans le claim `roles` du JWT. Il existe deux rôles dans la configuration JWT du projet Startix: `ROLE_ADMIN` et `ROLE_USER`

Ce code définit un bean Spring qui personnalise la manière dont les rôles (authorities) sont extraits d'un token JWT pour les utiliser dans les annotations de sécurité comme @PreAuthorize.

```
@Bean
public JwtAuthenticationConverter jwtAuthenticationConverter() {
    JwtGrantedAuthoritiesConverter converter = new JwtGrantedAuthoritiesConverter();
    converter.setAuthorityPrefix(""); // Par défaut, Spring ajoute le préfixe "SCOPE_" ou "ROLE_" aux rôles. I
    converter.setAuthoritiesClaimName("roles"); //Indique que les rôles sont stockés dans le claim "roles" du

    JwtAuthenticationConverter jwtConverter = new JwtAuthenticationConverter();
    jwtConverter.setJwtGrantedAuthoritiesConverter(converter);
    return jwtConverter;
}
```

5. Utiliser @PreAuthorize dans les contrôleurs (ou services)

L'annotation @PreAuthorize dans Spring Security sert à restreindre l'accès à une méthode (généralement dans un contrôleur ou un service) en fonction des rôles ou des permissions de l'utilisateur.

Exemple de l'utilisation de cette annotation dans le NoteController.java

```
/**
 * GET /api/notes - Récupère toutes les notes
 */
```

```
@GetMapping
@PreAuthorize("hasAnyAuthority('ROLE_ADMIN', 'ROLE_USER')") // Autoriser l'accès pour le ROLE_ADMIN et le ROLE
public ResponseEntity<List<NoteDto>> getAllNotes() {
    return ResponseEntity.status(HttpStatus.OK).body(noteService.findAll());
}

/**
 * DELETE /api/notes/{id} - Supprime une note
 */
@DeleteMapping("/{id}")
@PreAuthorize("hasAuthority('ROLE_ADMIN')") // Autoriser l'accès seulement pour le ROLE_ADMIN
public ResponseEntity<Void> deleteNote(@PathVariable UUID id) {
    boolean deleted = noteService.deleteById(id);
    if (deleted) {
        return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
    }
}
```

6. Utiliser le TrustStore RTE

Afin d'établir une connexion avec GAIA, il est nécessaire d'utiliser le fichier de certificat trustStore de RTE, disponible à l'emplacement suivant : `/backend/src/main/resources/security/cacerts` dans le JAVA runtime. Pour cela, vous avez le choix entre deux méthodes.

Méthode-1: Placer le TrustStore dans JAVA_HOME

Placer le certificat trustStore `cacerts` situé dans `/backend/src/main/resources/security/cacerts` dans le répertoire d'exécution de Java `${JAVA_INSATLL_PATH}/lib/security`.

```
# Obtenir le chemin d'installation de Java
dzdo update-alternatives --config java

# Définir le chemin de votre JDK (exemple)
JAVA_INSATLL_PATH=/home/CHANGE_ME/jdk-21.0.1
```

Copier le fichier `cacerts` dans le dossier `lib/security` de Java :

```
mv ${JAVA_INSATLL_PATH}/lib/security/cacerts ${JAVA_INSATLL_PATH}/lib/security/cacerts_old
cp /backend/src/main/resources/security/cacerts ${JAVA_INSATLL_PATH}/lib/security/cacerts
```

NOTE: Cette méthode est recommandée pour les developpements en local

Méthode-2: Définir l'option Java `javax.net.ssl.trustStore`

Définir le trustStore via l'option Java javax.net.ssl.trustStore :

```
JAVA_OPTS=-Djavax.net.ssl.trustStore=src/main/resources/security/cacerts
```

Démarrer l'application en utilisant la variable `${JAVA_OPTS}`.

Un exemple avec Maven :

```
mvn spring-boot:run -Dspring-boot.run.jvmArguments="${JAVA_OPTS}"
```

Un exemple avec Java :

```
java ${JAVA_OPTS} -jar target/CHANGE_ME.jar
```

NOTE: Cette méthode est utilisée pour l'environnement Docker (voir le fichier Dockerfile)

Voici le script qui permet de générer un nouveau fichier trustore cacert contenant le certificat Root CA du serveur GAIA :

```
keytool -import -trustcacerts -alias "Autorite_Bureau_RTE_Autorite_Racine_RTE.cer" -file Autorite_Bureau_RTE_A
keytool -import -trustcacerts -alias "Autorite_Racine_RTE.cer" -file Autorite_Racine_RTE.cer -keystore cacerts
keytool -import -trustcacerts -alias "OPF_Autorite_Bureau.cer" -file OPF_Autorite_Bureau.cer -keystore cacerts
keytool -import -trustcacerts -alias "OPF_Autorite_Racine_RTE.cer" -file OPF_Autorite_Racine_RTE.cer -keystore
keytool -import -trustcacerts -alias "opf_rte-france_com.cer" -file opf_rte-france_com.cer -keystore cacerts -
```

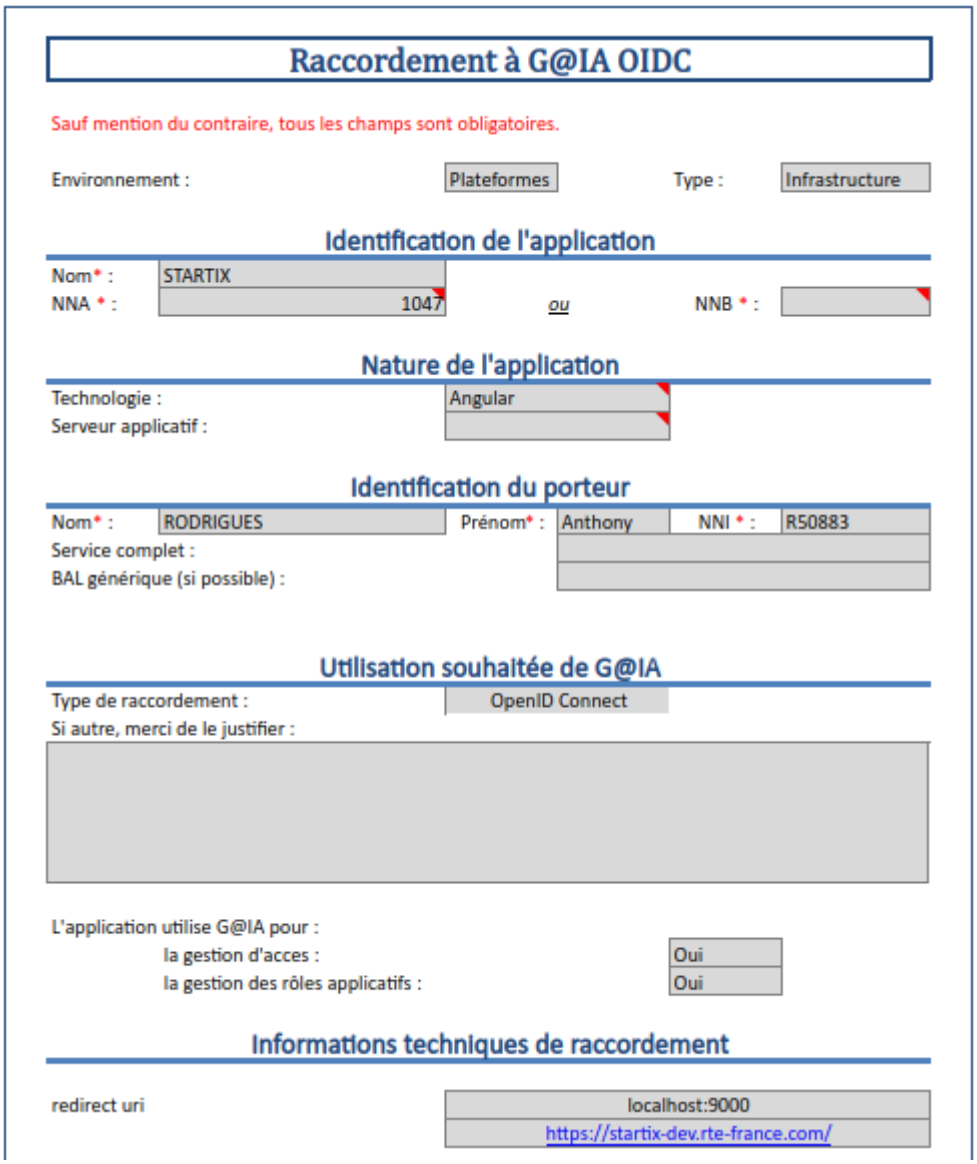
Voici la ressource pour les Autorité de Certification RTE SNP et OPF et la cacerts java associée.

Administratif

Procédure de demande Client OAuth2 (Ticket Cosmos)

1. **Créer un ticket Cosmos** pour demander un client OAuth2 dans G@IA/Ping Federate via le lien suivant : Demande de raccordement d'une application à la fédération d'identité G@IA.
2. La demande est a formuler avec une fiche Excel qui se trouve sous l'espace sharepoint `GAIA > OpenID Connect`: Fiche raccordement G@IA OIDC - V1.1.xlsx
3. Remplir les informations du ticket comme détaillé dans la section suivante.
4. **Préciser obligatoirement** dans la description ou les commentaires de votre ticket les informations techniques OIDC qui ne sont pas des champs standards du formulaire :
5. L'application utilise un **access token**.
6. Vous demandez un **claim** "**role**" dans l'access token pour la gestion des autorisations.
7. Les rôles/profils requis pour l'application (par exemple : `ROLE_ADMIN`, `ROLE_USER`).

Informations à fournir dans le ticket



Raccordement à G@IA OIDC

Sauf mention du contraire, tous les champs sont obligatoires.

Environnement :	Plateformes	Type :	Infrastructure

Identification de l'application

Nom* :	STARTIX		
NNA * :	1047	ou	NNB * :

Nature de l'application

Technologie :	Angular
Serveur applicatif :	

Identification du porteur

Nom* :	RODRIGUES	Prénom* :	Anthony	NNI * :	R50883
Service complet :					
BAL générique (si possible) :					

Utilisation souhaitée de G@IA

Type de raccordement :	OpenID Connect
Si autre, merci de le justifier :	

L'application utilise G@IA pour :

la gestion d'acces :	Oui
la gestion des rôles applicatifs :	Oui

Informations techniques de raccordement

redirect uri	localhost:9000
	https://startix-dev.rte-france.com/

Initialisation à la création

Note :
La MCO G@IA peut initialiser pour vous l'état de votre application lors de sa création. Cette section est facultative, et les actions correspondantes peuvent être réalisées ultérieurement sur demande.

Profils de l'application à gérer dans G@IA : Oui

Si oui, renseigner le tableau ci-dessous a titre informatif :

Attention : La granularité de la gestion de rôle via GAIA est préconisée à maximum 25. Au-delà de ce nombre, merci de configurer vos applications pour une gestion des rôles autonome.

Nom du profil	Description
STARTIX-ADMINS	Groupe des administrateurs de l'application

La création et suppression de role est gérée au travers des articles COSMOS suivant :
Plateformes : OPF - Modification configuration LDAP Plateformes
Production : GAIA - Modification configuration LDAP

NNI du ou des administrateurs G@IA de l'application (de préférence au moins un) :

R50883, R41619, MB74298T, R11461, R35294

La gestion des administrateurs delegués est gérée au travers des articles COSMOS suivant :
Plateformes : OPF - Demande du droit Administrateur d'une application dans GAIA en Plateforme
Production : GAIA - Demande du droit Administrateur dans GAIA (Production)

Pour un traitement d'habilitation par lot nous avons besoin des NNI et Nom du role (ou Nom d'affichage).

Exemple :

NNI des utilisateurs à habiliter initialement (un par cellule, ou fichier joint) :

R50883	STARTIX-ADMINS		
R41619	STARTIX-ADMINS		
MB74298T	STARTIX-ADMINS		
R11461	STARTIX-ADMINS		
R35294	STARTIX-ADMINS		

Pour toute question,
merci d'ouvrir une demande d'information et plannifer une reunion avec la BAL: RTE-DSIT-MCO-GAIA avec le numero de DT dans l'objet :

Plateformes : OPF - Demande d'information GAIA OPF
Production : GAIA - Demande d'information MCO GAIA

Formulaire de demande de raccordement d'une application à la fédération d'identité G@IA
Voici un guide pour remplir les champs du ticket Cosmos, basé sur les informations de l'image fournie.

Environnement

- **Environnement** : Plateformes
- **Type** : Infrastructure

Identification de l'application

- **Nom*** : Le nom de votre application (ex: `STARTIX`)

- **NNA*** : Le NNA de votre application (ex: `1047`)
- **NNB*** : (Optionnel) Le NNB de votre application

Nature de l'application

- **Technologie** : La technologie principale de votre application (ex: `Java`, `Angular`, `React`)
- **Serveur applicatif** : Le serveur applicatif si applicable.

Utilisation souhaitée de G@IA

- **Type de raccordement** : `OpenID Connect`
- **L'application utilise G@IA pour** :
- **l'authentification des utilisateurs** : Oui
- **le contrôle d'accès à l'application** : Oui
- **la gestion des rôles applicatifs** : Oui

Remarque : Il n'est pas forcément nécessaire de répondre oui à chaque question, ça dépend de votre choix de conception.

Informations techniques de raccordement

- **Base url** : L'URL de base de votre application pour l'environnement de développement (ex: `startix-dev.rte-france.com`)
- **Attributs utilisateur supplémentaires** : Si vous avez besoin d'attributs en plus de `mail`, `nom`, `prenom`, spécifiez-les ici. Par exemple, le `NNI`.

> **Note importante** : Pour une application frontend (comme Angular), les champs `Entity id` et `Assertion Consumer Service URL` sont généralement liés à SAML et peuvent ne pas être requis pour une configuration OIDC pure. C'est G@IA qui vous fournira les URLs de redirection à configurer.

En plus de ces champs, il est crucial de préciser les points suivants dans votre demande, comme déjà mentionné :

- **Type de Client** : `Public` (pour une application frontend s'exécutant dans le navigateur).
- **Méthode d'Authentification** : `PKCE` (Proof Key for Code Exchange) est obligatoire pour les clients publics.
- **Grant Type** : `Authorization Code`.
- **Refresh Token** : Demander l'activation du refresh token pour permettre le renouvellement automatique des tokens d'accès.
- **URLs de redirection (Redirect URIs)** : Les URLs vers lesquelles G@IA redirigera l'utilisateur après l'authentification.
- Pour le développement : `http://localhost:9000` (ou le port que vous utilisez)
- Pour la production : `https://votre-domaine-de-prod.com`
- **Scopes** : `openid profile email` sont les scopes standards.

Ce que vous recevrez

Après validation du ticket Cosmos, G@IA vous fournira :

1. **Client ID** (ex: `startix-dev-front-opf`)
2. **Issuer URL** (ex: `https://gaia-sso.opf.rte-france.com`)
3. **Endpoints OIDC** disponibles via `/.well-known/openid-configuration`
4. **Confirmation des rôles** configurés dans l'access token

Important : Les clients publics utilisant PKCE n'ont **pas de Client Secret**.

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	GAIA	Infrastructure	- https://cosmos.rte-france.com/esc?id=sc_cat_item&sys_id=7c9f14fff9414ed0af11820a23770b29

Documentation Centreon

À propos

La supervision applicative via le système permet de mesurer certaines métriques (CPU, RAM, Filesystem,) et la mise en place de sondes spécifiques afin d'alerter l'infogérant de certains dysfonctionnement. Au sein de RTE, cela est effectué par la solution CENTREON.

Tutoriel / Intégration

Centreon est disponible sur les environnements : - **OPF** maquettes (pour tester/developper..) https://centreon-mqt3.rte-france.com/centreon/ (et semi ancien https://centreon-mqt2.rte-france.com/centreon/) - **SNP 04.02.** Supervision Applicative système en SNP (CENTREON) url : https://supervsisnp.rte-france.com - **Horus 04.03.** Supervision Applicative système dans HORUS (CENTREON)

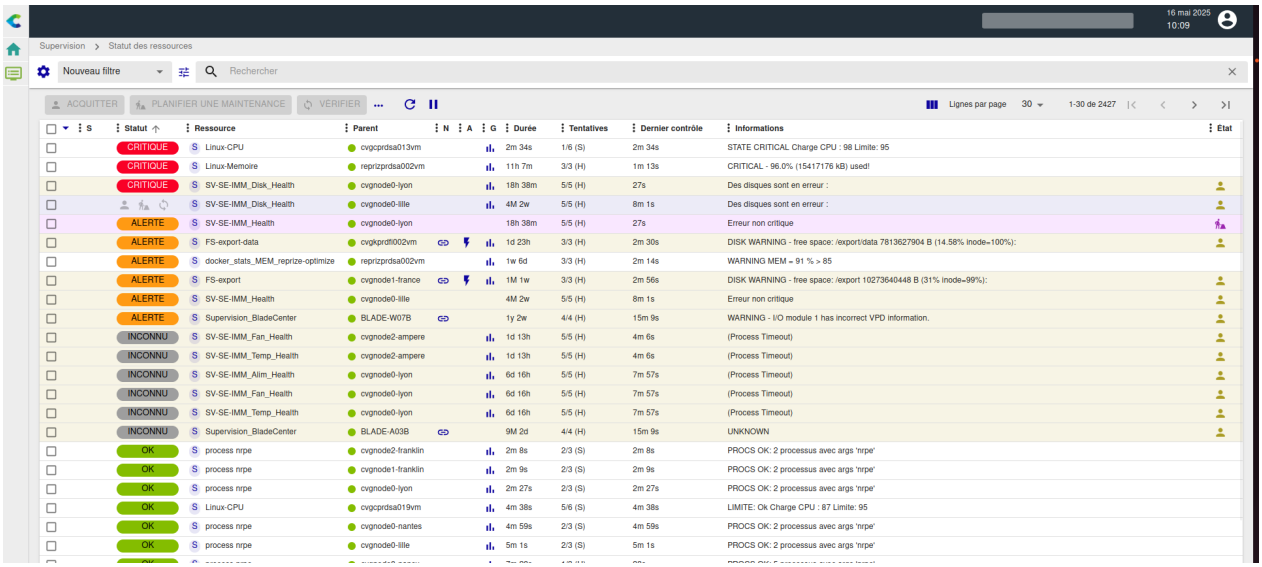
Le projet Startix utilise Centreon sur l'environnement **OPF**. Il est configuré pour superviser : - La base de données INPUT (PostgreSQL OPF) - L'infrastructure IK8S (Kubernetes) - URL API : Une API GET `/api/notes/count` qui renvoie le nombre des notes dans la base, avec une alerte automatique de type **WARNING** quand celui-ci dépasse un seuil de 5 notes et une alerte de type **CRITICAL** pour un nombre supérieur ou égal à 10 notes.

Remarque: Il faut savoir que toute demande de supervision projet en OPF sur Centreon n'est à la base que pour de la qualification. Une fois mise en place, personne ne s'occupe de la supervision à proprement dite et chaque supervision peut être désactivée/supprimée à tout moment en cas de besoin. Le CORS'N ne supervise pas l'OPF.

L'offre de service de Centreon est décrite sur DOKI.

SNP

Elle consiste en l'installation de sondes dont les résultats sont synthétisées sur une interface dont l'accès est commun à tout le pôle DDL.



Offre par défaut

Des sondes par défaut sont mises en place en fonction des technologies indiquées dans le DA.

Il existe des sondes propres à Linux, Windows, docker, etc. Toute cette offre par défaut est décrite sur DOKI. Pour exemple, pour une application dockerisée déployée sur une VM linux, on aura ces sondes installées par défaut :

Indicateur	Fréquence	Seuil d'alerte	Seuil critique	Commentaires	Sonde
Linux-FS	5 min	20%	10%	Surveillance de l'espace libre dans un FS donné.	FS_<nom du FS>
Linux-FS /	5 min	20%	10%	Surveillance de l'espace libre dans le FS racine (/)	FS_-racine
Linux-FS /opt	5 min	15%	10%	Surveillance de l'espace libre dans le FS /opt	FS_opt
Linux-FS /opt/vtom	5 min	20%	10%	Surveillance de l'espace libre dans le FS /opt/vtom	FS-opt-vtom
Linux-FS /usr	5 min	10%	5%	Surveillance de l'espace libre dans le FS /usr	FS_usr
Linux-FS /var/	5 min	25%	20%	Surveillance de l'espace libre dans le FS /var	FS_var
Linux-FS /var/log	5 min	20%	10%	Surveillance de l'espace libre dans le FS /var/log	FS-var-log
Linux-FS /exploit	5 min	40%	30%	Surveillance de l'espace libre dans le FS /exploit	FS_exploit
Linux-FS /home	5 min	15%	10%	Surveillance de l'espace libre dans le FS /home	FS_home
Linux-FS /tmp	5 min	20%	10%	Surveillance de l'espace libre dans le FS /tmp	FS_tmp

Indicateur	Fréquence	Seuil d'alerte	Seuil critique	Commentaires	Sonde
Docker mémoire	5 min	80%	90%	Consommation mémoire de l'image docker	docker_stats_MEM_<image>
Docker CPU	5 min	80%	90%	Consommation CPU de l'image docker	docker_stats_CPU_<image>
Docker Status	5 min		Image docker pas au statut Running	Vérifie que l'image docker est bien au statut running	docker_status_<image>

Il existe aussi des sondes disponibles pour la supervision Kubernetes IK8S. Voici quelques métriques Clés à Superviser :

Les métriques ci-dessous nous à superviser nous ont été fournis par l'équipe PEPITE :

Métrique	Description	Seuil
Pods en état "Pending"	Nombre de pods en attente de ressources.	Alerte à partir de **3 pods pendant 5 minutes**.
Pods en état "CrashLoopBackOff"	Nombre de pods échouant à démarrer.	Alerte dès qu'au moins **1 pod est concerné**.
Pods en état "ErrImagePull"	Problèmes liés aux images Docker.	Alerte immédiate.
Nombre total de pods par node	Charge sur chaque nœud.	Alerte si > 90% de la capacité maximale.
Utilisation de la mémoire par node	Mémoire utilisée par rapport à la capacité.	Alerte si > 80% pendant 5 minutes.
Utilisation du CPU par node	CPU utilisé par rapport à la capacité.	Alerte si > 80% pendant 5 minutes.
Nombre de nodes indisponibles	Nodes marqués comme "NotReady".	Alerte dès qu'un nœud est concerné.
Latence API Kubernetes	Temps de réponse des requêtes vers l'API.	Alerte si > 1 seconde pendant 1 minute.

Sondes spécifiques

Pour l'installation de sondes spécifiques, il faut se rapprocher de l'équipe DESIGN responsable de l'offre de service (contact **design-osa-supervision.dsit@rte-france.com**) en amont pour en vérifier la faisabilité.

Voici une liste non exhaustive des sondes qu'il est possible d'installer :

- Requête SQL (uniquement disponible sur base Hexa)
- Apparition de mots clefs dans les logs (attention : l'alerte est levée si de nouveaux logs sans ces mots clefs sont enregistrés)
- Présence ou Absence d'un service applicatif
- Présence / taille / âge d'un fichier
- Nombre de fichiers dans un répertoire
- Réponse à une URL
- Présence d'une chaine de caractères dans une page web
- La récurrence d'un mot sur une page
- Mémoire consommée pour un processus spécifique
- CPU consommé pour un processus spécifique
- Etc.

Pour chaque sonde, il faudra préciser : - Le seuil d'alerte - La fréquence de surveillance - optionnellement : l'action CORSN à effectuer en cas d'alerte. Si celle-ci est non spécifiée, le CORSN contactera l'équipe projet en cas d'alerte.

Remarque : Si vous activez la supervision de la base de données via Centreon, il est impératif de prévoir un nombre suffisant de connexions disponibles dans le pool de connexion de la base de données. Centreon utilise des connexions pour interroger les métriques, et une saturation du pool pourrait entraîner des erreurs de supervision ou des ralentissements.

En production: Une fois la faisabilité validée avec DESIGN, il est nécessaire de mettre ajour le DSBE en explicitant les besoins de supervision dans la partie adéquate et s'il s'agit d'une demande de supervision spécifique, il faut joindre la PTI que l'équipe DESIGN fourni en cas de besoin.

Après avoir modifié le DSBE, il faut le mettre à jour dans Doki et faire une demande COSMOS de "mise en cohérence du DSBE" (avec la PTI en PJ) qui prendra en compte les demandes ajoutées dans le DSBE pour que le CORSN installe ces sondes..

Le CORSN mettra un DEX dans Doki qui indiquera ce qui a été mise en place conformément à la demande exprimée dans le DSBE.

OPF

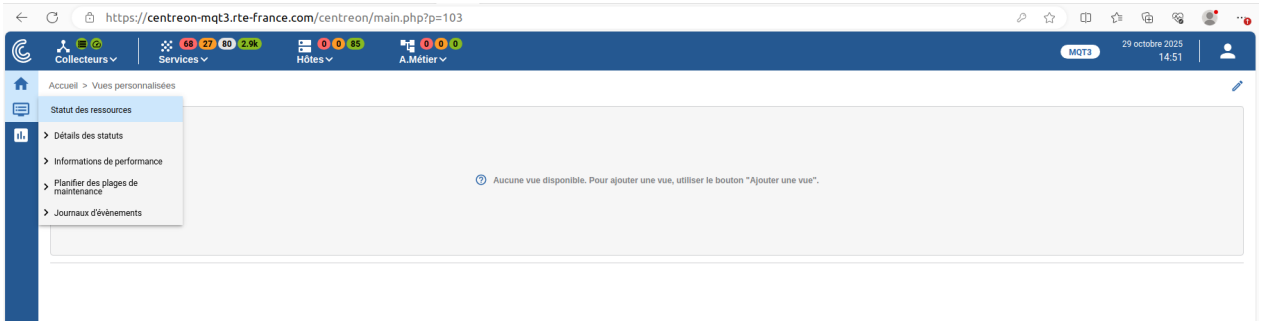
les centreon "maquette" mqt servent à developper/iterer/qualifier des sondes,

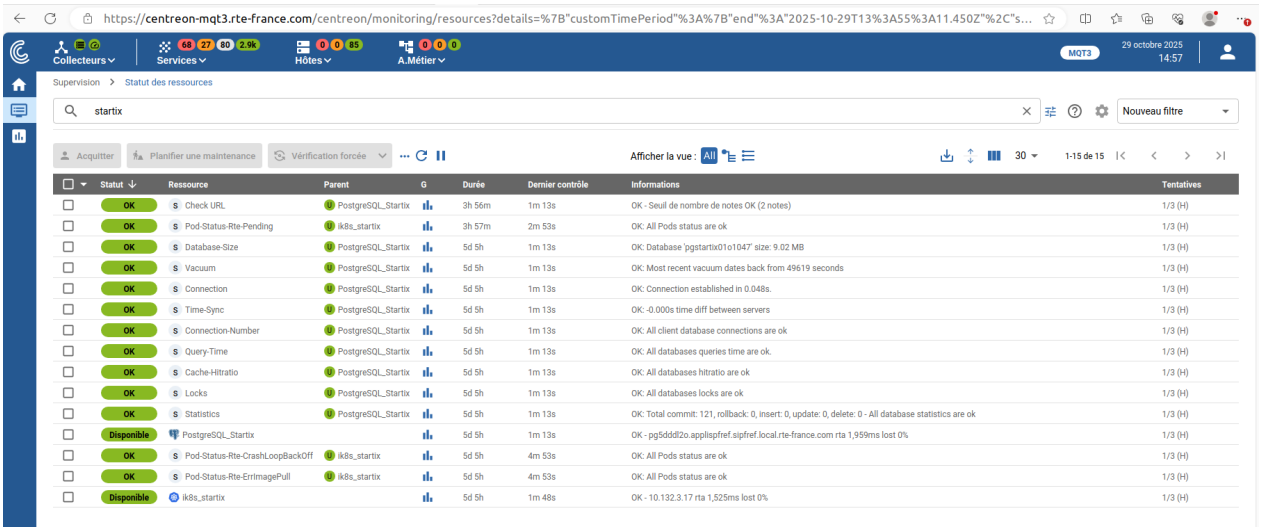
Il y a 3 centreon maquettes en opf, grosso modo les versions précédentes qui sont graduellement remplacée. Ils appellent ces centron des maquettes, d'où leurs noms: maquete1, maquete2, maquete3. - La maquete1 est cassée depuios longtemps et n'est plus utilisée, à oublier - mqt2 un peu vieux mais encore utilisée - mqt3 est actuellement (24/06/2025) celle à utiliser pour les nouveaux trucs.

Attention à ne pas confondre avec "le centreon opf infras" qui est pour opf mais n'est pas une maquette, sert à l'exploitation (comme si c'était de la prod https://centreon-opf.rte-france.com/centreon/) d'OPF, et avec l'ancien OPF qualification de plugin (https://supervapplisopf.rte-france.com/centreon/) mais maintenant ce travail a lieu sur les mqt (qui sera supprimé bientôt à date du 24/06/2025)

Pour avoir accès à centreon maquette et voir fonctioner et developper, faire un mail à **DSIT--DESIGN-OSA-SUPERVISION** design-osa-supervision.dsit@rte-france.com pour demander un acces readonly pour voir, et échanger avec la personne qui répond pour qu'elle fasse les créations/modifications des sondes à developper.

Voici un exemple de la maquette Startix sur centreon-mqt3.





Administratif

Toutes vos demandes ou toutes questions seront à adresser à la liste **DSIT--DESIGN-OSA-SUPERVISION** design-osa-supervision.dsit@rte-france.com

À propos

Checkstyle est un outil d'analyse statique open-source qui permet de : - Vérifier que le code Java respecte un ensemble de règles de style. - Détecter automatiquement les erreurs de formatage, les conventions de nommage, les imports inutiles, etc. - Appliquer des standards comme Google Java Style ou Sun Style, ou des règles personnalisées.

Utile pour maintenir une base de code cohérente et lisible dans les équipes de développement.

Tutoriel / Intégration

1. Ajouter Checkstyle dans un projet Maven

- Ajouter le plugin dans pom.xml

```
<plugin>
    <artifactId>maven-checkstyle-plugin</artifactId>
    <version>${maven-checkstyle-plugin.version}</version>
    <executions>
        <execution>
            <id>validate</id>
            <phase>validate</phase>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <configLocation>checkstyle.xml</configLocation>
```

```
        <includeTestSourceDirectory>true</includeTestSourceDirectory>
        <consoleOutput>true</consoleOutput>
        <failsOnError>true</failsOnError>
    </configuration>
</plugin>
```

- Ajouter la version du plugin dans la balise `<properties>` du `pom.xml`

```
<maven-checkstyle-plugin.version>3.5.0</maven-checkstyle-plugin.version>
```

- Ajouter le fichier checkstyle.xml dans le projet (vérifier l'emplacement configuré dans le plugin du pom.xml)
- Exécuter Checkstyle avec Maven

```
mvn clean verify
```

Le build échouera si des violations sont détectées.
Un rapport HTML est généré dans target/site/checkstyle.html.

2. Intégrer Checkstyle dans IntelliJ IDEA

Installer le plugin CheckStyle-IDEA

- Aller dans `File > Settings > Plugins` (ou IntelliJ IDEA > Preferences > Plugins sur macOS)
- Rechercher puis installer le plugin `CheckStyle-IDEA`
- Restart IDE.

Configurer Checkstyle dans IntelliJ

- Aller dans `File > Settings > Tools > Checkstyle`
- Cliquer sur le bouton +pour ajouter une configuration
- Donner un nom à la configuration
- Sélectionner le fichier `checkstyle.xml`
- Sélectionner le scope `all`
- Cocher la case Active pour l'activer puis sauvgarder

Lancer une inspection

- Aller dans `View > Tool Windows > Checkstyle`
- Sélectionner la configuration ajoutée
- Cliquer sur Check pour lancer l'analyse

À propos

Cette documentation décrit le processus de mise en place de l'intégration continue d'une application sur le DevOps RTE, en utilisant GitLab et Jenkins.
L'objectif est de mettre en place d'un pipeline CI/CD efficace.

Tutoriel / Intégration

L'intégration continue à RTE est effectuée par la plateforme Devin IC, qui est une instance Jenkins déployée pour le DevOps RTE.
Jenkins s'appuie sur des pipelines "as code" définis dans des fichiers Jenkinsfile.

La mise en place de liens entre les événements du cycle de vie du code (MR, release, push) et l'exécution des pipelines s' appuie sur le système des webhooks GitLab.

Le backend/Jenkinsfile définit un pipeline pour la construction et l'analyse du projet Startix Backend SpringBoot.
Un deuxième frontend/Jenkinsfile définit un pipeline pour la partie Frontend Angular.

Voici une explication concise de son fonctionnement :

Architecture

Pipelines

Le pipeline s'exécute sur un nœud Jenkins mutualisé (node('build')), et suit ces étapes :

1. Pipeline Backend Java (startix-back)

🦊 **Checkout**

- Récupère le code depuis Git et détermine la branche active.

🏭 **Build mvn**

- Build l'application en utilisant le maven

⚗️ **Unit Test**

- Lance les tests unitaires et les tests d'intégrations

🔬 **Code Quality**

- Lance l'analyse SonarQube du dépôt Git.

🐳 **Build docker**

Cette étape, cruciale pour la livraison, n'est exécutée que si le paramètre `doitConstruireDocker` est activé.

- Construction de l'image : Une image Docker est construite à partir du fichier /backend/Dockerfile.
- Nommage de l'image : L'image est taguée avec le nom `startix-inca-app/startix-startix-back` et la version du build.
- Publication sur INCA : Si la construction réussit, l'image sera poussée vers le registre Docker privé de RTE (INCA) si le paramètre `doitLivrer` est activé.

💥 **Gestion des erreurs et nettoyage**

- Capture et affiche les erreurs avec des notifications par email vers le maintainer définit dans le Jenkinsfile.
- Nettoie le workspace à la fin.

2. Pipeline Frontend Angular (startix-front)

🦊 **Checkout**

- Récupère le code depuis Git et détermine la branche active.

🚧 **npm install**

- Télécharge les dépendances du package.json.

🏭 **Build npm**

- Build l'application en utilisant le NPM.

⚗️ **Unit Test**

- Lance les tests unitaires.

🔬 **Code Quality**

- Lance l'analyse SonarQube du dépôt Git.

🐳 **Build docker**

Cette étape, cruciale pour la livraison, n'est exécutée que si le paramètre `doitConstruireDocker` est activé.

- Construction de l'image : Une image Docker est construite à partir du fichier /frontend/Dockerfile.
- Nommage de l'image : L'image est taguée avec le nom `startix-inca-app/startix-startix-front` et la version du build.
- Publication sur INCA : Si la construction réussit, l'image sera poussée vers le registre Docker privé de RTE (INCA) si le paramètre `doitLivrer` est activé.

💥 **Gestion des erreurs et nettoyage**

- Capture et affiche les erreurs.
- Nettoie le workspace à la fin.

Résumé : Ces pipelines automatisent le build, les tests unitaires et integrations, avec une gestion des paramètres et des erreurs.

Webhook et lancement manuel

Les **webhooks GitLab** et le **lancement manuel d'un job Jenkins** sont deux façons très courantes de démarrer un pipeline Jenkins, mais ils répondent à des besoins différents.

Webhook GitLab

- Un webhook permet à GitLab d'informer Jenkins automatiquement lorsqu'un événement se produit (ex. un push sur une branche, une création de merge request...).
- Pour configurer un Webhook, voir la section Configuration Webhook

Lancement manuel d'un job Jenkins

- Il est possible aussi de démarrer un job Jenkins manuellement depuis l'interface Jenkins (Voir Lancement manuel d'un job jenkins).
- Le lancement manuel du job jenkins permet d'avoir un contrôle total du moment du déclenchement, utile pour les jobs ponctuels et parfait en cas de test ou débogage.

Déclaration des constantes et paramètres dans le Jenkinsfile

- Voici la liste des variables clés à modifier dans le Jenkinsfile dans la section CONSTANTES DU JOB :
  - NOM_PROJET : Nom du projet / application.
  - BRANCHE_PRINCIPALE : Branche principale sur Git.
  - MVN_VERSION : version du maven.
  - CURRENT_JDK : version du jdk.
  - NOM_COMPOSANT : Nom du module applicatif concerné.
  - NOM_PROJET_INCA : Nom du projet sur la plateforme INCA (registre Docker).
  - NOM_DOCKER_IMAGE : Nom de l'image Docker générée.
  - SLAVE_DOCKER : Label de l'agent Jenkins utilisé pour les opérations Docker.
  - VERSION_QUALIFIER : Qualifier utilisé dans le nom de version. Les versions sont taggées au format ..-build* => ex : 01.00.00-build1
  - NODE_JS_VERSION : version du node pour le frontend.
- Depuis l'interface Jenkins, ajouter et configurer les paramètres qui existent dans la section ## PARAMETRES DEPUIS Jenkins ## du fichier Jenkinsfile (voir Ajouter un paramètre du job Jenkins).
- Voici la liste des paramètres du JOB Jenkins :
  - VERSION_APP : versionApp (String), Version de l'image Docker en cours de construction.
  - PREPARER_LIVRAISON : doitLivre (Booléen), Si coché, déclenche les étapes de livraison complètes (build Docker, push, etc.).
  - DOIT_TAGGER : doitTagger (Booléen), Si coché, déclenche la création d'un tag Git.
  - Construction docker : doitConstruireDocker (Booléen), Détermine si l'image Docker doit être construite. Cette option est automatiquement activée si PREPARER_LIVRAISON ou DOIT_TAGGER est coché.

Prérequis

1. Demander un NNA

Si le projet n'a pas encore un NNA, Demander un NNA - Architecture SI - GOPro Collaboratif (rte-france.com)

Information Startix	Valeur
NNA	1047
Nom court	Startix

2. Création d'un espace GitLab et Jenkins

Un espace GitLab et Jenkins est requis pour mettre en place l'intégration continue du projet.

Merci de consulter la section dédiée listant les demandes à effectuer pour bénéficier de ces services.

3. Mise en place d'un Slave Jenkins (si besoin)

Note : Cette étape peut être ignorée si le projet ne repose pas sur Docker.

Dans le cas contraire, merci de se référer à la section dédiée.

4. Structure du Repository GitLab

Une fois l'espace GitLab créé, l'utilisateur doit :

- Créer un projet dans cet espace GitLab. Voir la section dédiée.
- Pousser les fichiers de son projet.
- Pousser un fichier Jenkinsfile qui contient la définition du pipeline d'intégration continue.

Par exemple, pour un projet java :

```
├── src : Code de l'application
├── Jenkinsfile
├── Dockerfile
└── pom.xml
```

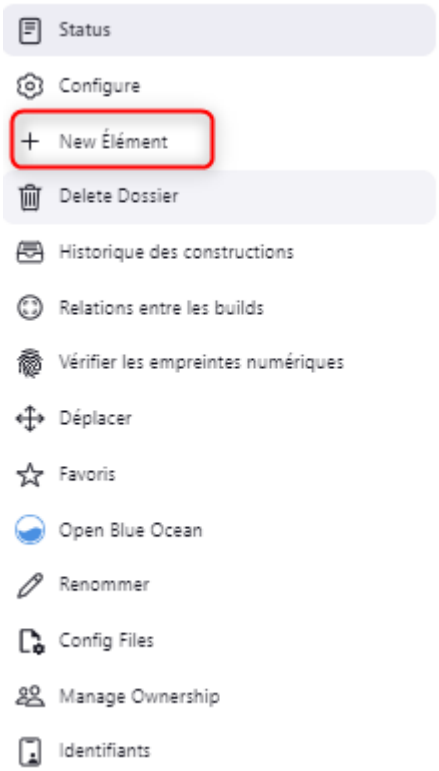
Exemple, pour un projet node :

```
├── src : Code de l'application
├── Jenkinsfile
├── Dockerfile
└── package.json
```

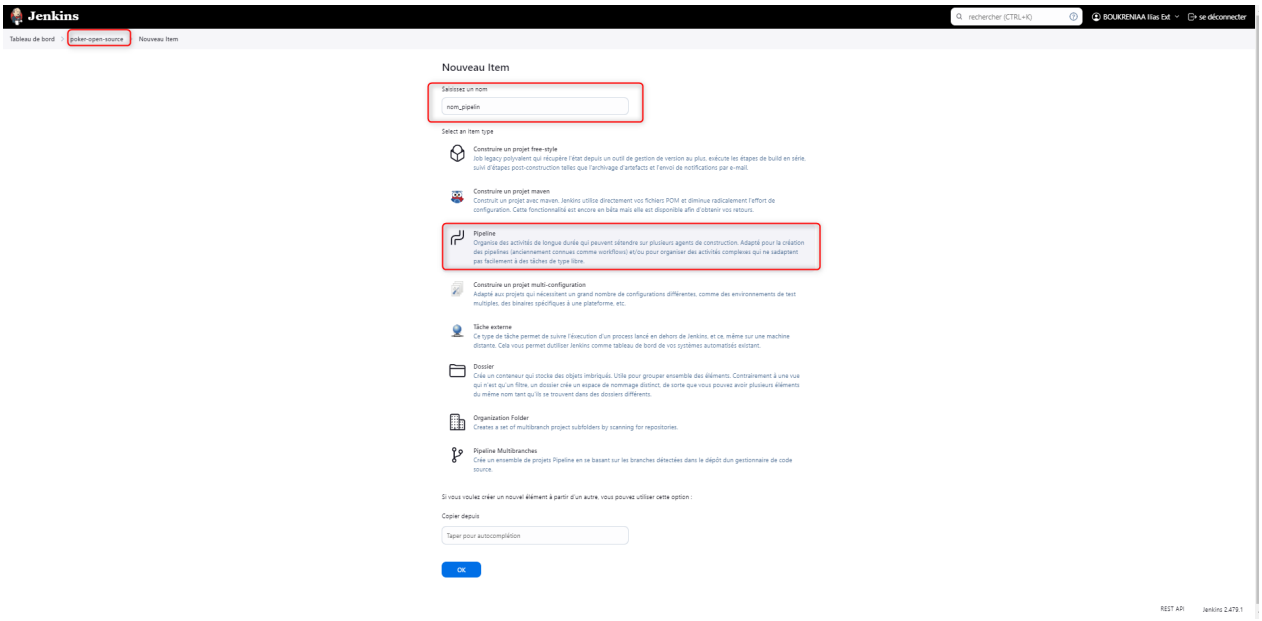
1. Action Jenkins

1.1 Création et Configuration de la Pipeline

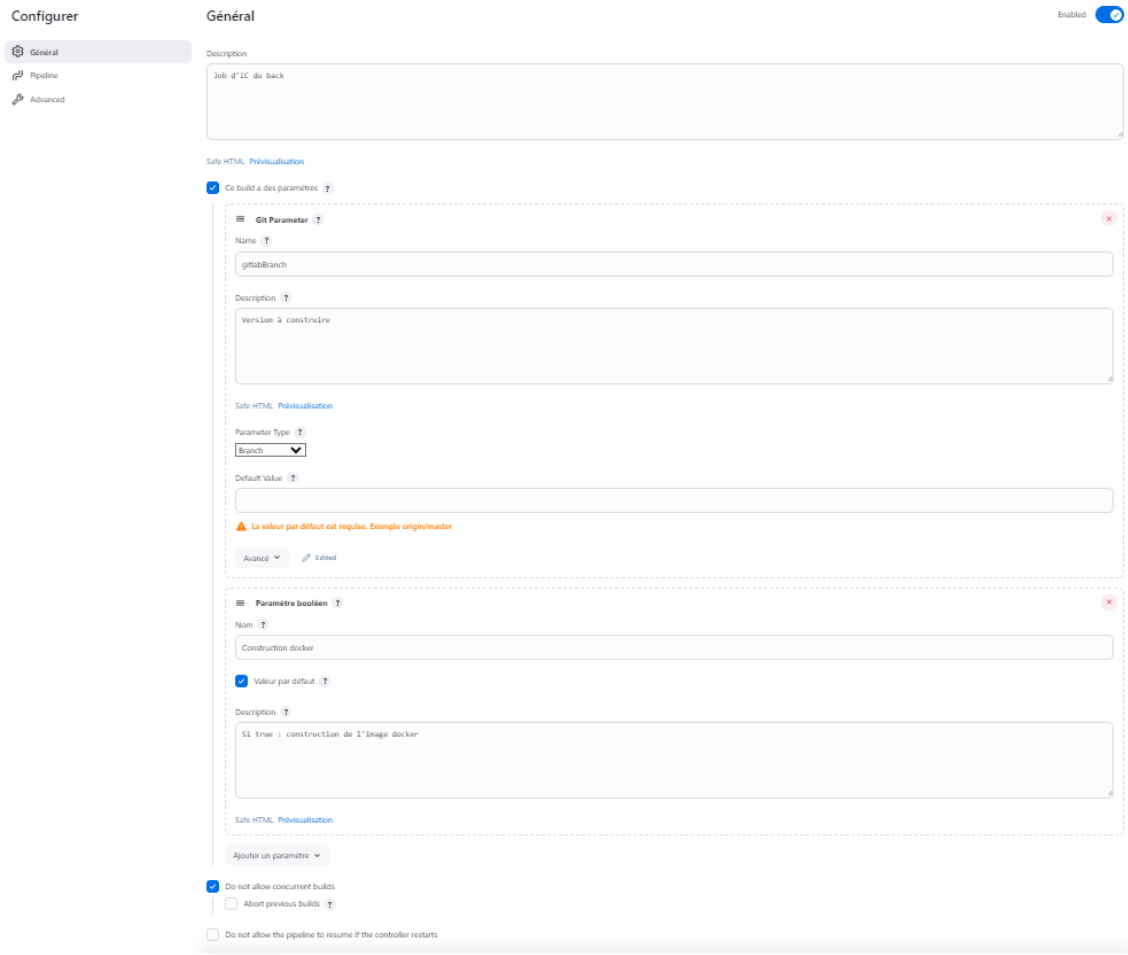
- Se positionner sur l'espace Jenkins créé par l'équipe Devin : https://devin-ic.rte-france.com/
- Cliquer sur New Item

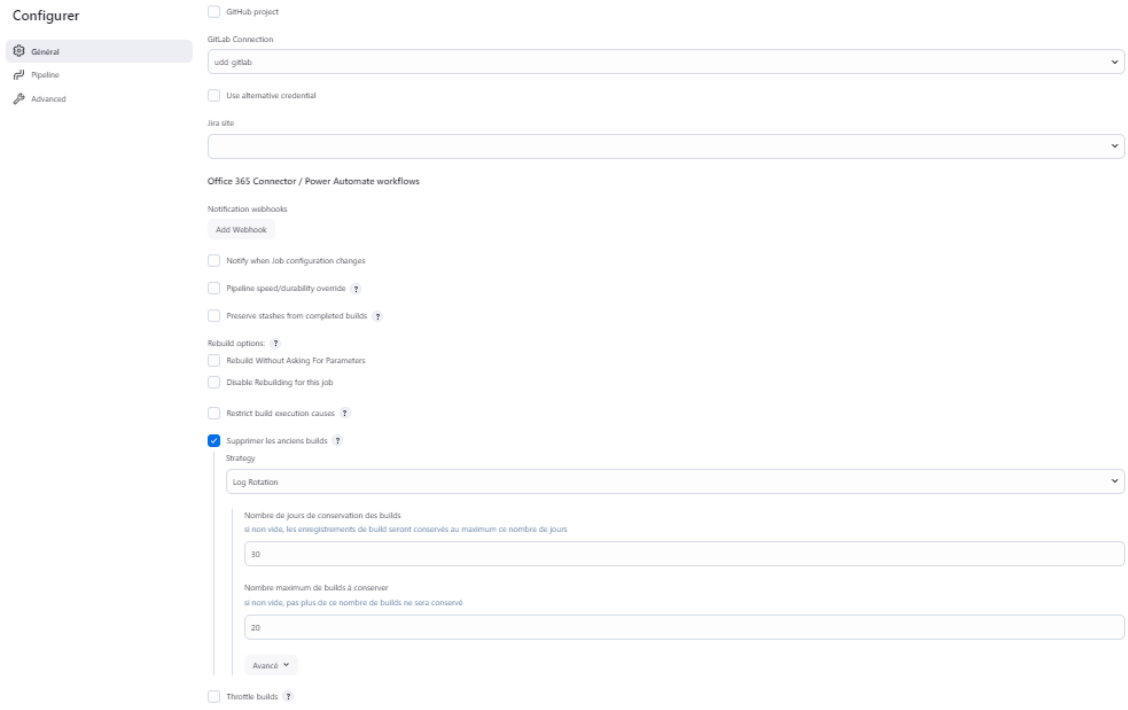


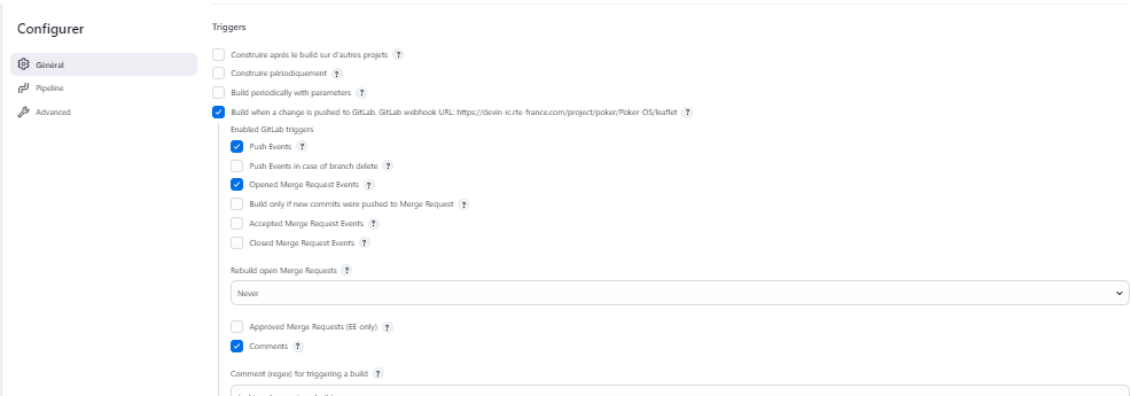
- Saisir un nom de pipeline



• Configurer la pipeline en renseignant le Jenkinsfile


Configurer
Général
Pipeline
Advanced
Général
Enabled
Description
Job d'IC du back
Safe HTML Prévisualisation
Ce build a des paramètres
Git Parameter
Name
gitlabBranch
Description
Version à construire
Safe HTML Prévisualisation
Parameter Type
Branch
Default Value
La valeur par défaut est requise. Exemple origin/master
Avancé
Edited
Paramètre booléen
Nom
Construction docker
Valeur par défaut
Description
Si true : construction de l'image docker
Safe HTML Prévisualisation
Ajouter un paramètre
Do not allow concurrent builds
Abort previous builds
Do not allow the pipeline to resume if the controller restarts


Configurer
Général
Pipeline
Advanced
GitHub project
GitLab Connection
udd gitlab
Use alternative credential
Jira site
Office 365 Connector / Power Automate workflows
Notification webhooks
Add Webhook
Notify when Job configuration changes
Pipeline speed/durability override
Preserve stashes from completed builds
Rebuild options:
Rebuild Without Asking For Parameters
Disable Rebuilding for this job
Restrict build execution causes
Supprimer les anciens builds
Strategy
Log Rotation
Nombre de jours de conservation des builds
si non vide, les enregistrements de build seront conservés au maximum ce nombre de jours
30
Nombre maximum de builds à conserver
si non vide, pas plus de ce nombre de builds ne sera conservé
20
Avancé
Throttle builds


Configurer
Général
Pipeline
Advanced
Triggers
Construire après le build sur d'autres projets
Construire périodiquement
Build periodically with parameters
Build when a change is pushed to GitLab. GitLab webhook URL: https://devin-ic.rte-france.com/project/poker/Poker OS/leaflet
Enabled GitLab triggers
Push Events
Push Events in case of branch delete
Opened Merge Request Events
Build only if new commits were pushed to Merge Request
Accepted Merge Request Events
Closed Merge Request Events
Rebuild open Merge Requests
Never
Approved Merge Requests (EE only)
Comments
Comment (regex) for triggering a build

Secret Token dans Jenkins

Le Secret Token situé dans la Section "Trigger > Avancée > Secret Token" doit être sauvegardé, car il sera utilisé dans la configuration du webhook de GitLab pour déclencher automatiquement le pipeline.

Il faut donc l'introduire dans le champ prévu à cet effet lors de la configuration du webhook sur GitLab (voir section Action Gitlab).

1.2 Ajouter un paramètre du job Jenkins

Dans le fichier Jenkinsfile, il existe une section ## PARAMETRES DEPUIS Jenkins ## contenant les paramètres du job Jenkins.

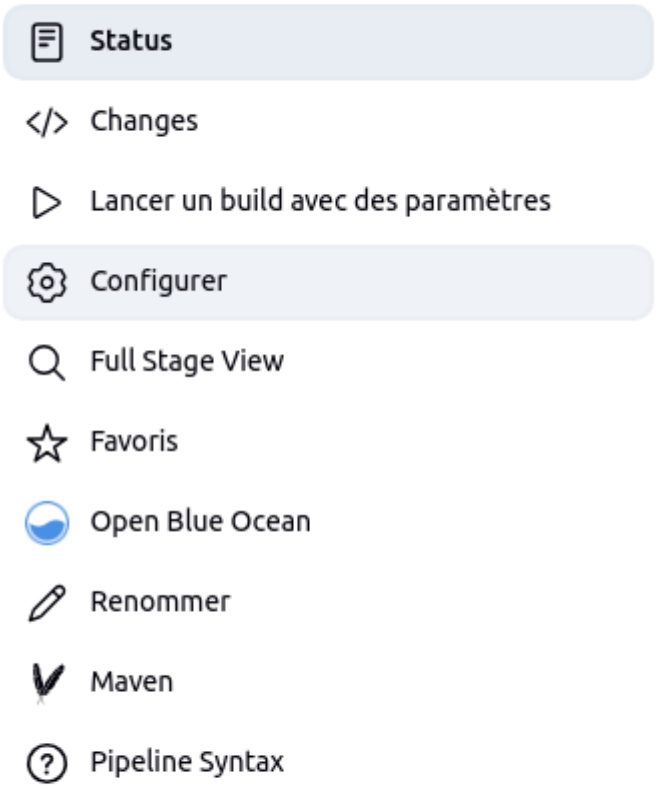
```
// exemple d'un parametre de type boolean "IS_RELEASE" (par défaut = false)
Boolean isRelease = utils.readBoolean(params.IS_RELEASE, false)
```

Pour accéder à la valeur d'un parametre dans le Jenkinsfile, utiliser :

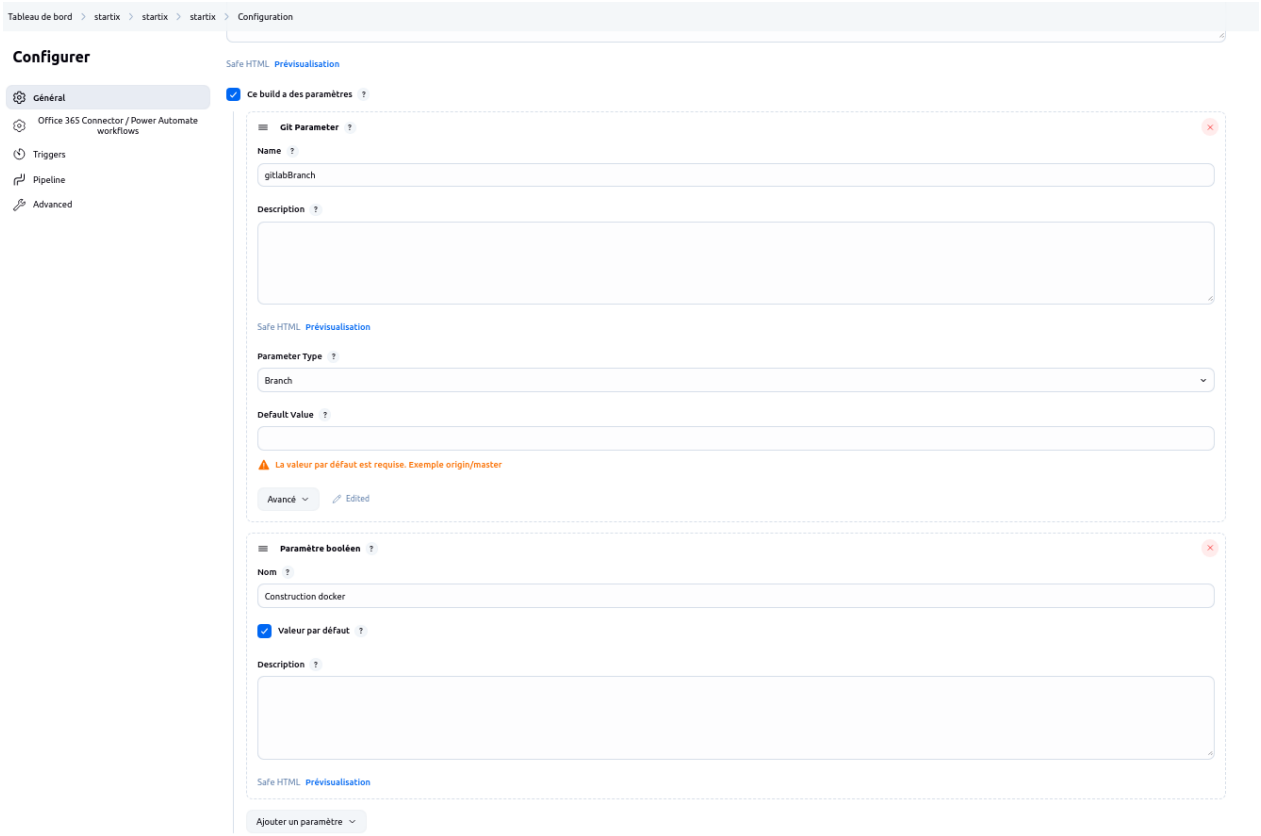
```
params.PARAM_NAME
```

Pour ajouter un paramètre via l'interface Jenkins :

- Aller sur la page du Job
- Cliquer sur "Configurer"



- Cocher "Ce build a des paramètres" puis cliquer sur "Ajouter un paramètre"



- Choisir le type du paramètre :
    - String Parameter
    - Boolean Parameter
    - Choice Parameter
    - File Parameter (etc.)
- Renseigner le nom, la description et les valeurs par défaut
- Enregistrer

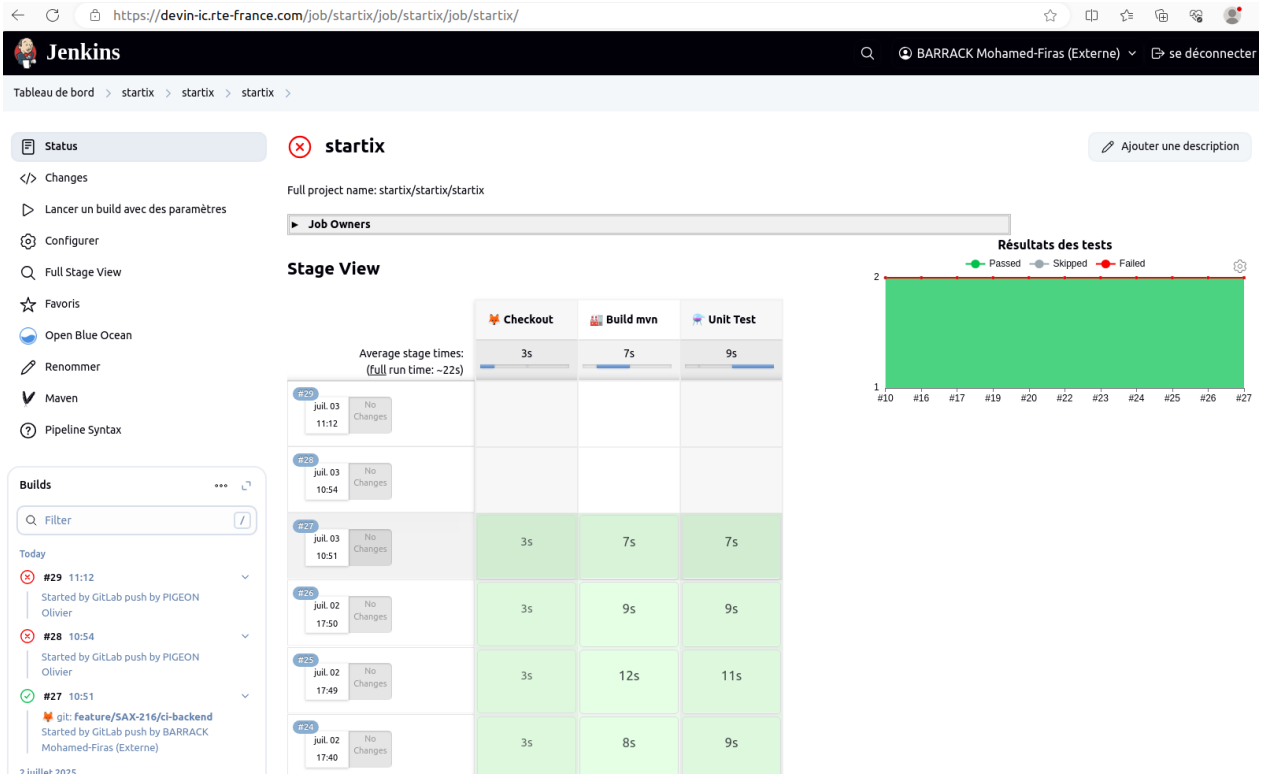
Priorité entre Jenkinsfile et configuration UI :
Voici un résumé du comportement de Jenkins concernant les valeurs par défaut des paramètres dans les jobs :

- Lors du lancement manuel du job, Jenkins proposera un formulaire pour remplir les paramètres et les valeurs saisies écrasent celles par défaut du Jenkinsfile.
- Quand un job Jenkins est déclenché automatiquement (via webhook), le Jenkins va utiliser la valeur par défaut définie dans la configuration du job (UI) — et non celle du Jenkinsfile.
- Si le paramètre n'a pas été défini dans l'interface Jenkins (UI), dans ce cas, sa valeur par défaut dans le Jenkinsfile est respectée.

Situation du paramètre	Valeur par défaut utilisée par Jenkins
Définie uniquement dans le `Jenkinsfile`	Valeur du `Jenkinsfile`
Définie dans l'interface Jenkins (UI)	Valeur de l'UI (elle **remplace** celle du `Jenkinsfile`)
Définie dans les deux (UI + Jenkinsfile)	Jenkins utilise **celle de l'UI**
Définie dans le Jenkinsfile mais absente dans l'UI	Valeur du `Jenkinsfile` utilisée

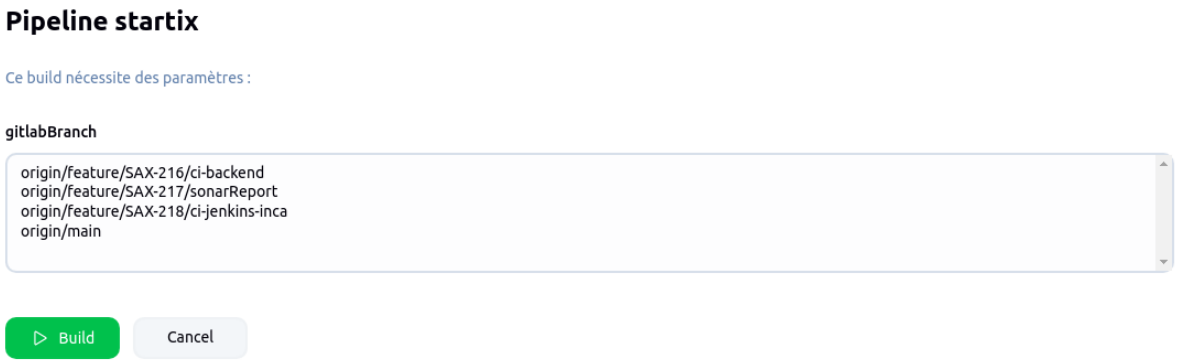
1.3 Lancement manuel d'un job Jenkins

- Depuis l'interface du job Jenkins, cliquer sur "Lancer un build avec des paramètres"



- Choisir la branche sur laquelle faire tourner le pipeline en sélectionnant la valeur souhaitée du paramètre " gitlabBranch".

Noter que l'utilisation d'un paramètre de type Git branch a permis de lister automatiquement les branches disponibles dans l'IHM limitant les risques d'erreur de saisie.

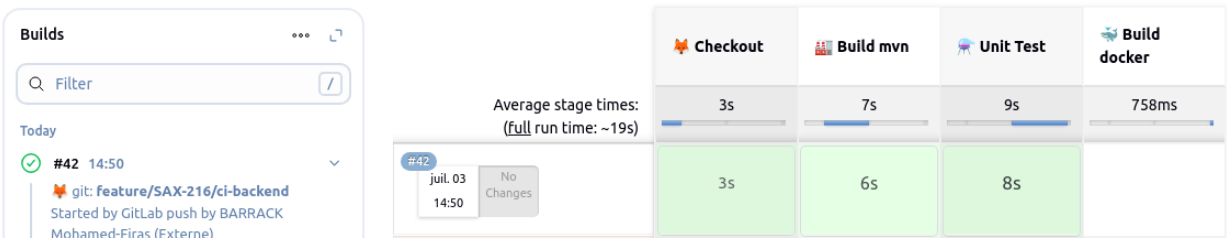


1.4 Sortie de la console d'un job Jenkins (logs)

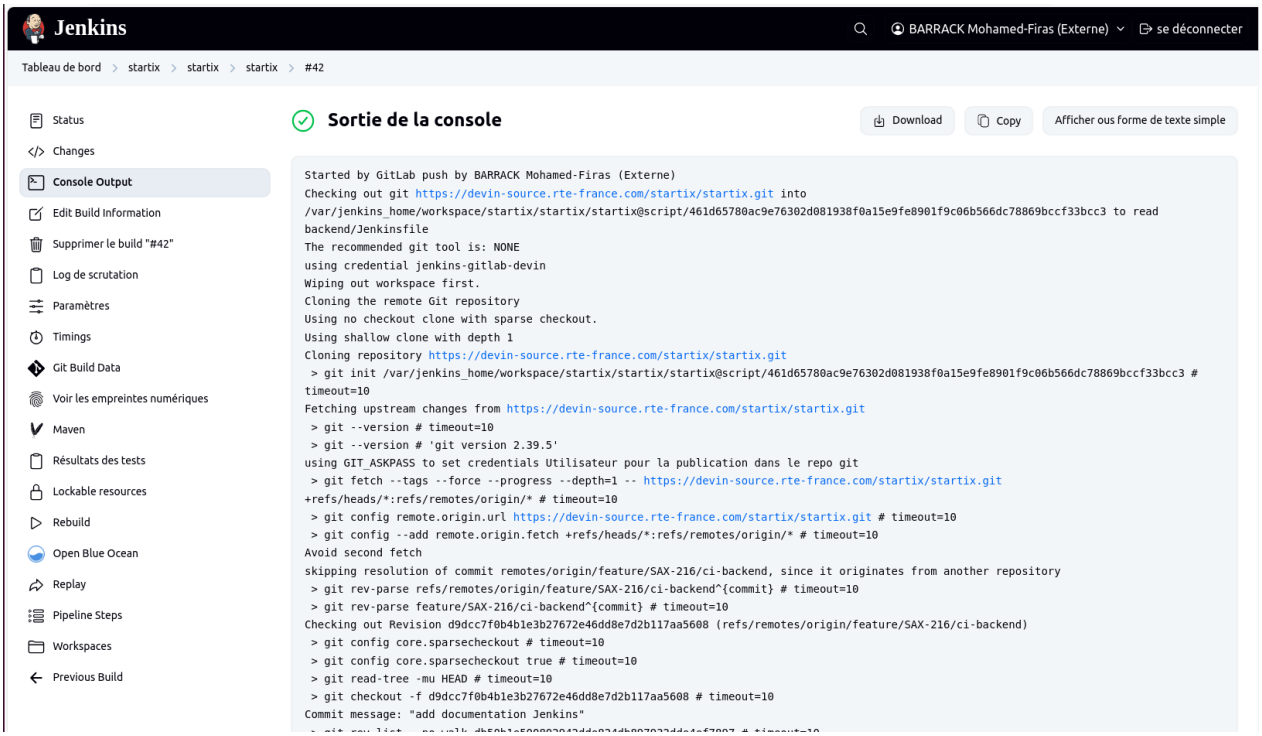
La sortie de la console (ou "console output") d'un job Jenkins est un journal des logs de l'exécution du job.
Il contient tous les messages générés pendant que Jenkins exécute le pipeline, ligne par ligne, étape par étape.

Pour voir la sortie de la console du job :

- Cliquer sur le numéro du job (ex : #42).



- Dans la page du job, cliquer sur "Console Output".



Les logs détaillés du job seront alors affichées, ligne par ligne, depuis le démarrage jusqu'à la fin.

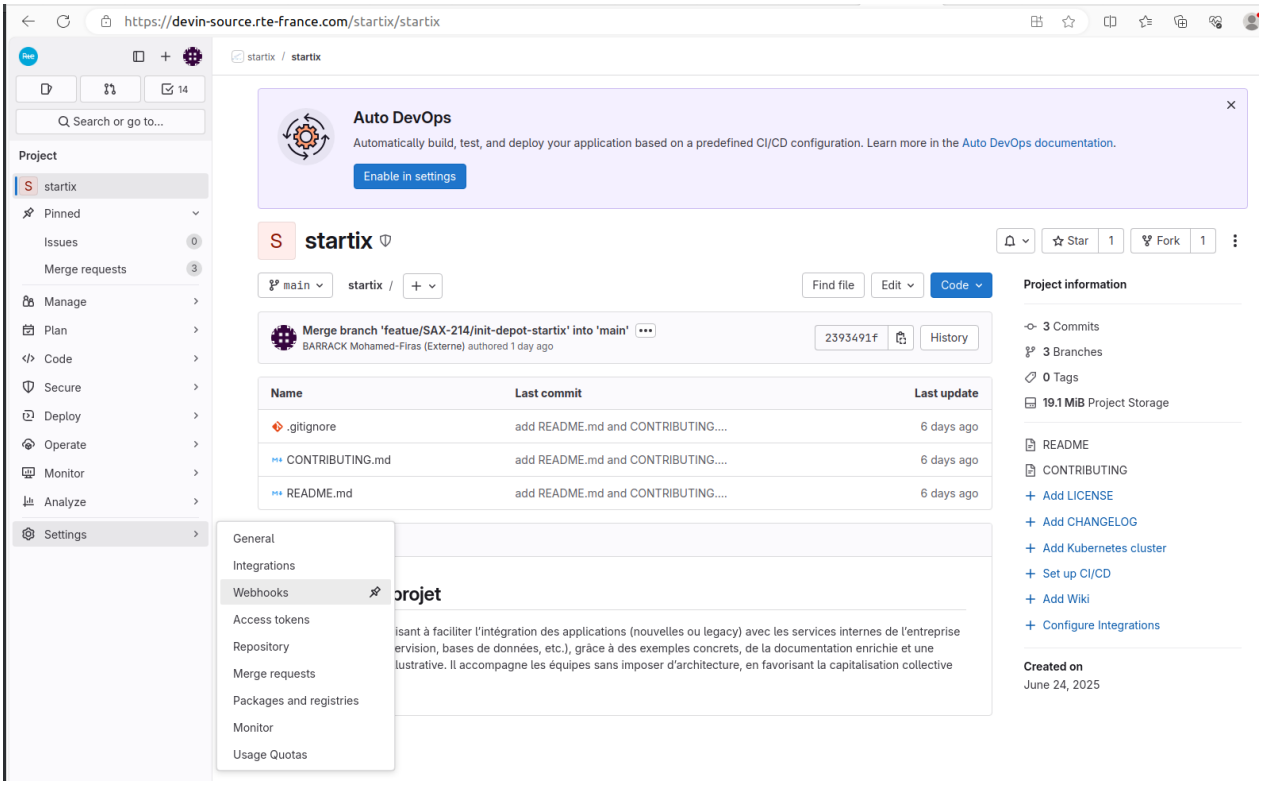
2. Action Gitlab

1.1 Configuration Webhook

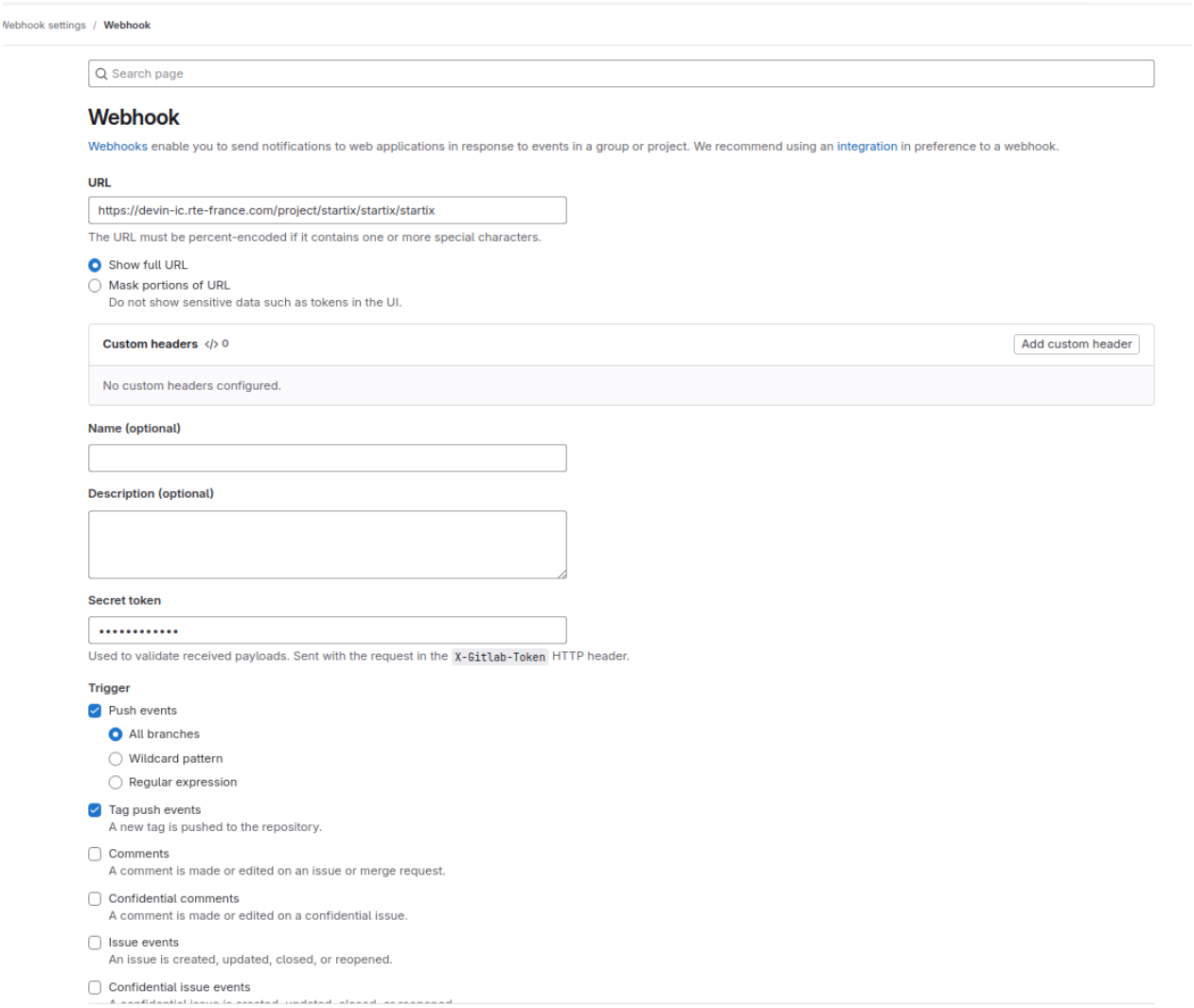
Un **webhook** doit être mis en place pour déclencher automatiquement le pipeline Jenkins lors de la création d'un tag sur GitLab.

Procédure :

- Aller sur GitLab > Settings > Webhooks



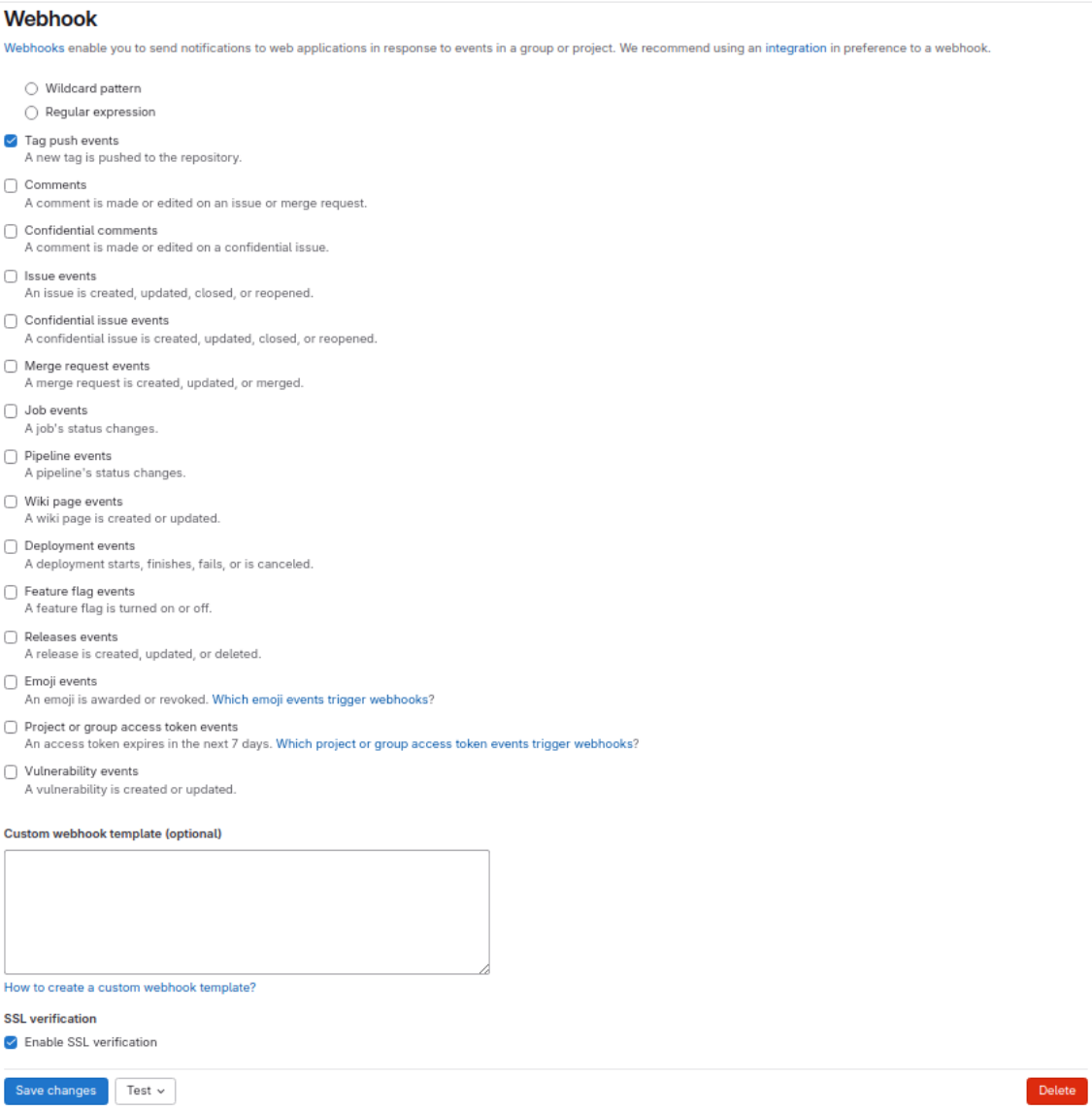
- Ajouter un nouveau Webhook



- **URL** du projet dans Jenkins.
- **Secret token** nécessaire à la configuration sera fourni lors de la création du pipeline (voir section Action Jenkins).
- **Triggers** permettent de déclencher automatiquement des jobs ou pipelines en réponse à des événements externes (comme un push Git, une pull request, etc.)

Exemples:

Trigger Webhook	Événement déclencheur	Exemple d'usage
Push Git	Commit ou push sur une branche	Déclenche un build à chaque mise à jour du code
Pull Request / Merge Request	Création ou mise à jour d'une PR	Lance des tests pour valider la PR automatiquement
Tag / Release	Création d'un tag ou d'une release	Déclenche un déploiement ou l'exécution d'un pipeline de release



Administratif

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	Gitlab & Jenkins	Offre DevOps	https://gopro-tickets.rte-france.com/browse/DEVMCO-7488
D2	Jenkins	Offre DevOps	https://gopro-collaboratif.rte-france.com/pages/viewpage.action?pageId=100835369
D3	Gitlab	Offre DevOps	https://gopro-collaboratif.rte-france.com/pages/viewpage.action?pageId=102542061

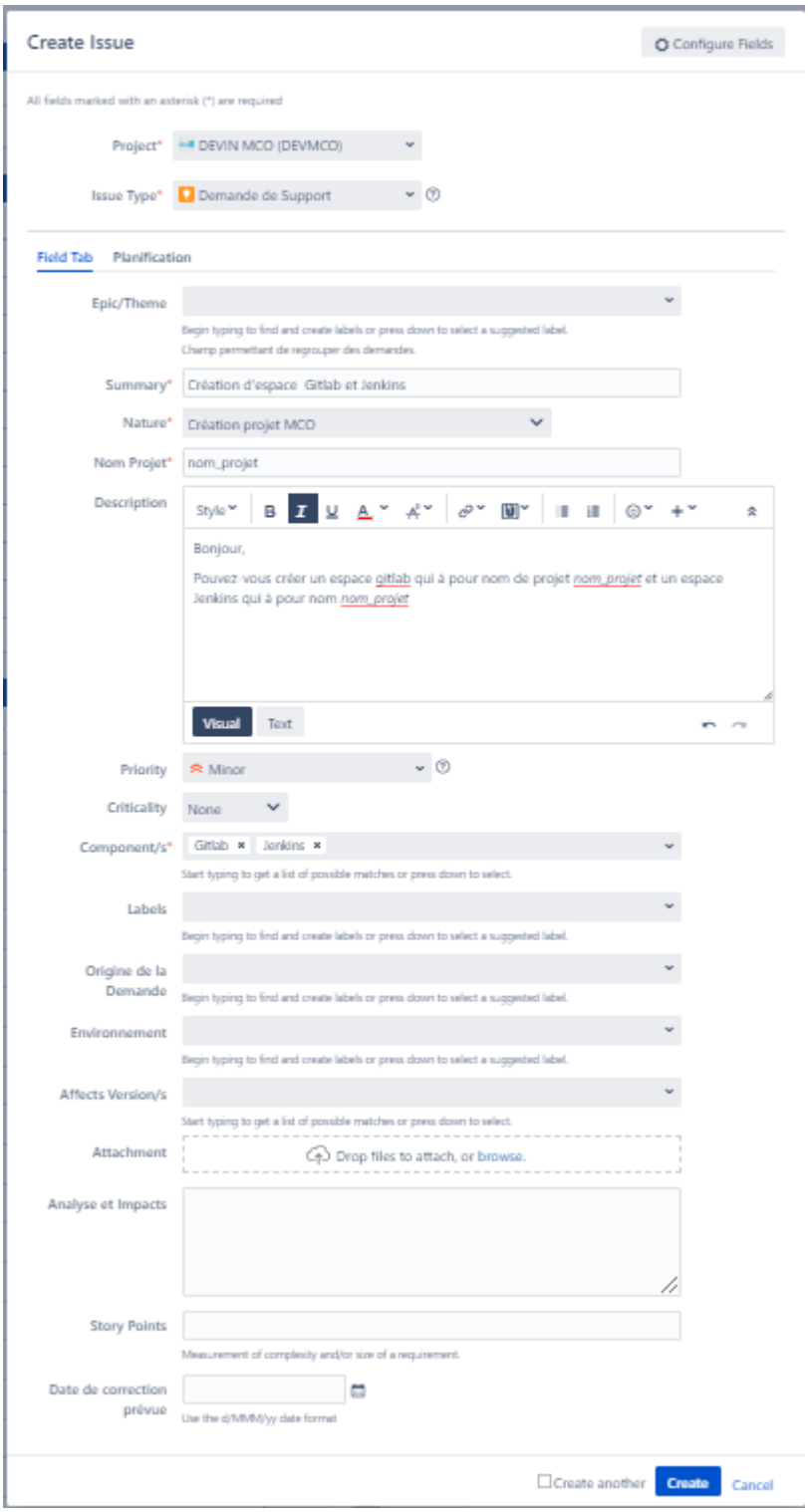
D1. Demande de création d'un espace GitLab et Jenkins

Il est nécessaire de faire une demande explicite à l'équipe Devin pour obtenir :

- Un espace GitLab
- Un espace Jenkins

Procédure :

Se rendre sur Jira et soumettre une demande de création d'espaces GitLab DEVMCO-7488 - Création d'un espace projet GitLab et Jenkins comme suit :



- date de la rédaction de la demande ci-dessus : 24/01/2025
- date de résolution de la demande : 29/01/2025

D2. Mise en place d'un Slave Jenkins (si besoin)

Lorsque le projet s'appuie sur Docker, il est impératif de mettre en place un slave Jenkins spécifique projet contenant Docker pour créer et builder l'image.

Le slave sera renseigné par la suite dans une fonction `dockerBuild` par le parametre `targetPtf` (voici un exemple de la fonction dockerBuild)

- Une documentation interne RTE explique la procédure d'installation. (Procédure d'installation d'un slave Jenkins)
- La version originale est adaptée pour **CentOS 7**, mais un repo pour **OL8** est disponible.

D3. Création d'un projet GitLab (Devin-Source)

Comment créer un projet GitLab et y ajouter des utilisateurs.

À propos

Docker est une plateforme de conteneurisation qui permet d'empaqueter une application avec toutes ses dépendances dans un conteneur léger, portable et isolé. Cela garantit que l'application fonctionne de manière cohérente quel que soit l'environnement (développement, test, production).

Dans le cadre de ce projet, Docker est utilisé pour : - Conteneuriser le projet Startix. - Intégrer le processus de build dans une pipeline Jenkins pour automatiser la livraison - Faciliter le déploiement sur les differents environnements.

Tutoriel / Intégration

Intégrer Docker dans un projet Java

1. Ajouter le fichier Dockerfile

Ajouter le fichier Dockerfile qui permet de construire une image docker pour JAVA.

Remarque: Le fichier Dockerfile utilise le TrustStore RTE à travers l'option JAVA `javax.net.ssl.trustStore` pour autoriser l'accés à l'authentification Oauth2 GAIA.

2. Builder et lancer l'image Docker en local

- Générer le fichier `.jar` de l'application

```
cd backend
mvn clean package
```

- Utiliser le fichier docker-compose pour lancer l'application

```
docker-compose up --build
```

Vérifier que l'application fonctionne : Accéder à http://localhost:8080

- Arrêter et supprimer tous les éléments créés par docker-compose up

```
docker-compose down
```

Ou

```
docker-compose down --volumes --rmi all
```

- --volumes : supprime les volumes nommés
- --rmi all : supprime les images créées par le build

Intégrer Docker dans un projet Node

1. Ajouter le fichier Dockerfile

Voici un exemple simple d'un Dockerfile qui permet de construire une image docker basé sur le NGINX.

```
# Serveur pour servir l'application construite
FROM inca.rte-france.com/rtsi/nginx:1.21.5-alpine3.15.0

# Ajouter la configuration nginx
COPY ./nginx.conf /etc/nginx/nginx.conf

# Supprime la page index nginx par defaut
RUN rm -rf /usr/share/nginx/html/*

# Copier les fichiers de build construits
COPY ./dist/startix-front/browser /usr/share/nginx/html

# Exposer le port nginx 8080
EXPOSE 8080

#ENTRYPOINT ["nginx", "-g", "daemon off;"]
ENTRYPOINT ["/bin/sh",  "-c", "( echo \"cat <<EOF\" ; cat /usr/share/nginx/html/assets/env.js.template ; echo
```

Ce Dockerfile permet de : - Configurer Nginx avec un fichier personnalisé nginx.conf. - Copier les fichiers fichiers générés par le build Angular dans le dossier web de Nginx. - Générer dynamiquement le fichier env.js à partir d'un template env.js.template pour injecter des variables d'environnement. - Lancer Nginx pour Docker.

2. Ajouter le fichier nginx.conf

- Ajouter le fichier de configuration nginx.conf au même niveau que le Dockerfile. Voici le contenu du fichier nginx.conf :

```
worker_processes 4;

events { worker_connections 1024; }

http {
    large_client_header_buffers  4 16k;
    server {
        listen 8080;
        root  /usr/share/nginx/html;
        include /etc/nginx/mime.types;

        location / {
            try_files $uri $uri/ /index.html =404;
        }
    }
}
```

3. Ajouter les assets

- Sous le dossier `src`, créer un dossier `assets` et ajouter les fichiers env.js et env.js.template.
- Voici le contenu du fichier env.js :

```
(function (window) {
  window["ENV"] = window["ENV"] || {};
  // env varibles section
  window["ENV"].VAR_1 = 'var1_value';
  window["ENV"].VAR_2 = 'var2_value';

})(this);
```

- Voici le contenu du fichier env.js.template :

```
(function (window) {
  window["ENV"] = window["ENV"] || {};
  // env varibles section
  window["ENV"].VAR_1 = '${VAR_1}';
  window["ENV"].VAR_2 = '${VAR_2}';

})(this);
```

Le `env.js` est utilisé pour injecter des variables d'environnement au moment du démarrage, sans avoir à reconstruire l'application.

Le Dockerfile génére dynamiquement le fichier env.js à partir du fichier env.js.template qui contient les variables d'environnement de l'application et leurs valeurs comme l'URL de l'API, la version, le mode debug, etc.

Ces variables seront définies dans `window`, qui est l'objet global dans un navigateur.

L'application peut ensuite lire ces valeurs via window.ENV.

En pratique, cela permet d'accéder à des variables dynamiques via window.ENV.VAR_1.

4. Attacher le script env.js dans le fichier index.html

- Pour charger les variables d'environnement dans l'objet global `window` du navigateur, il faut ajouter le script env.js dans le fichier index.html :

```
<!-- Load environment variables -->
<script type="application/javascript" src='./assets/env.js'></script>
```

5. Adapter le type de l'objet Window dans l'application

- Dans le typescript, il faut déclarer une interface pour l'object `ENV` qui contient les variables d'environnement dans l'objet global `window`. Voici les lignes à ajouter dans le fichier globals.d.ts sous le dossier `src` (au même niveau avec le fichier `index.html`):

```
// contenu du fichier globals.d.ts

interface Window {
  ENV: Env;
}

interface Env {
  // Globals Env
  VAR_1: string;
}
```

6. Utilisation de l'objet Window.ENV dans l'application Angular

- Pour utiliser les variables d'environnement dans le code Anguler, on utilise les fichiers environment.ts (pour l'env Prod) et environment.local.ts (pour l'env Local).
- Voici le contenu du fichier `environment.ts`:

```
const nativeWindowEnv = typeof window !== 'undefined' ? (window as Window).ENV : undefined;

export const environment = {
  var_1: nativeWindowEnv?.VAR_1 || 'default_var1_value',

  production: true,
};
```

Ce code permet de : - Centraliser la configuration dans un objet environment que l'application peut l'utiliser partout. - Récupérer dynamiquement les valeurs d'environnement depuis window.ENV, si elles existent, sinon on utilise des valeurs par défaut.

Exemple d'utilisation dans un fichier .ts:

```
// Importer la constante environment
import { environment } from '../environments/environment';

// Affiche la valeur injectée dans $VAR_1 ou 'default_var1_value'
console.log(environment.var_1);
```

7. Builder et lancer l'image Docker en local

- Générer le build `dist` de l'application

```
cd frontend
npm run build
```

- Construire l'image Docker

```
cd frontend
docker build -t nom-image:version .
```

Modifier le `nom-image` et la `version`.

Exemple: `docker build -t startix-front:0.1.0-SNAPSHOT .`

- Lancer le conteneur Docker

```
docker run -p 9000:8080 nom-image:version
```

- Ou, lancer le conteneur Docker avec les fichiers d'envrionnement dans env.conf et private.conf:

```
cd frontend
cat env.conf private.conf > temp.env
docker run --env-file temp.env -p 9000:8080 nom-image:version
rm temp.env
```

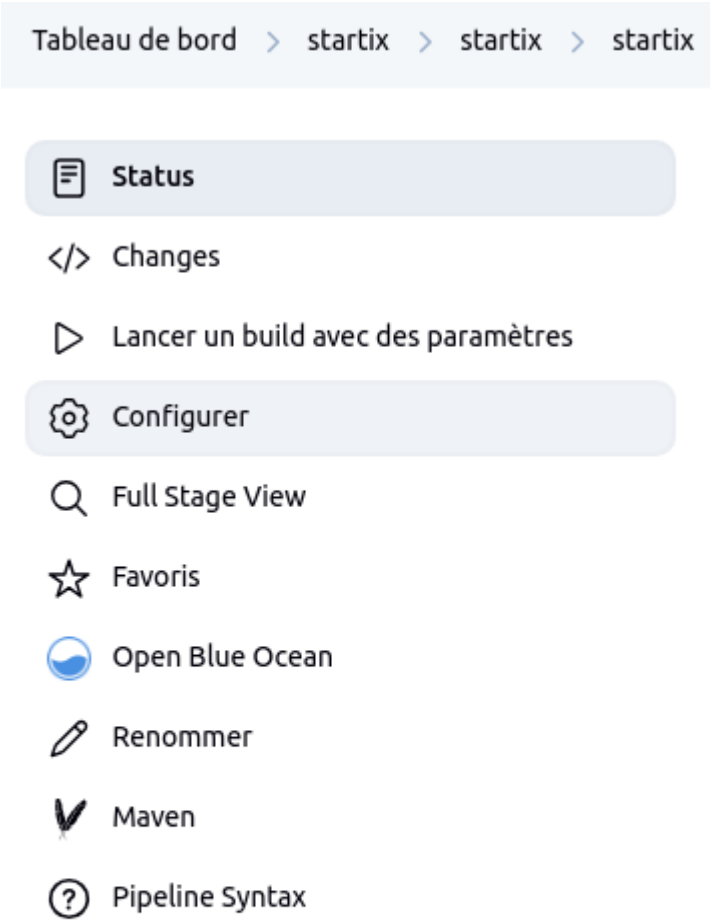
- Vérifier que l'application fonctionne : Accéder à http://localhost:9000

Intégration dans Jenkins

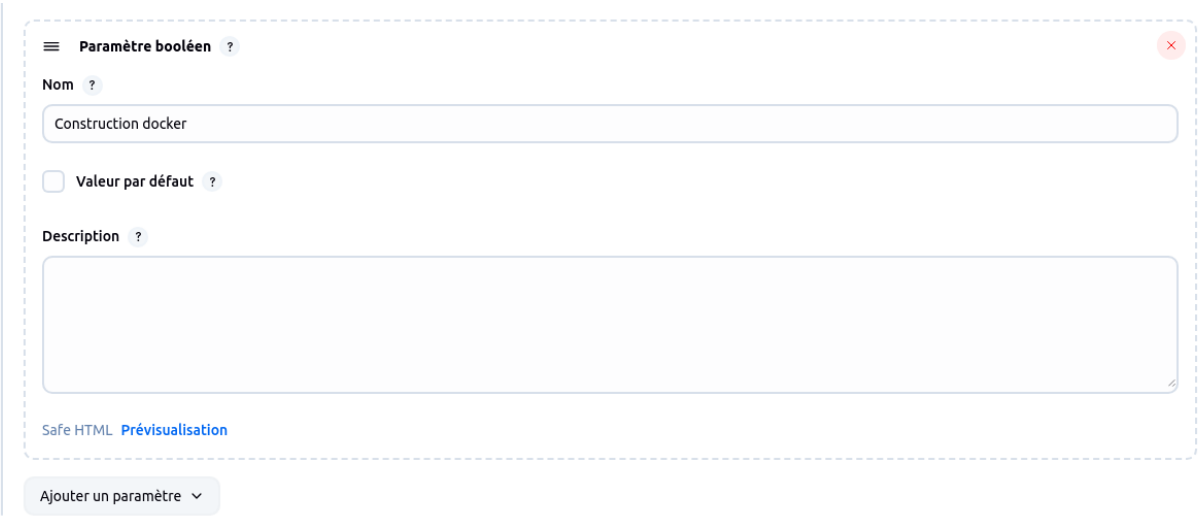
L'étape BuildDocker dans le Jenkinsfile automatise la construction et le push de l'image Docker.

Pour activer l'étape de Build Image Docker, il faut ajouter le parametre boolean `doitConstruireDocker` dans la configuration de la pipeline.

Voici les étapes à suivre: - Aller sur la configuration du job jenkins du projet



- Ajouter un paramètre de type `paramètre choix` avec le nom `Construction docker` et laisser la valeur par défaut à false (case décoché).



- Pour la partie Backend, adapter le code de l'étape `Build docker` dans le /backend/Jenkinsfile

```
stageDevin ('🐳 Build docker', doitConstruireDocker) {
    echo "🐳 Construction Docker"
    dir('backend') {
        dockerBuild {
            dockerContext = 'Dockerfile,target/*.jar'
            repo = NOM_PROJET_INCA
            loginInca = 'startix-inca-app'
            imgName = NOM_DOCKER_IMAGE
            versionImg = buildVersion
            targetPtf = SLAVE_DOCKER
            push = doitLivrer
            prefixRegistry = true
            keepBuiltImage = false
            options = ['build-arg': ["VCS_REF=${gitResult.GIT_COMMIT}", "VCS_URL=${gitResult.GIT_URL}"]]
        }
    }
}
```

- Pour la partie Frontend, adapter le code de l'étape `Build docker` dans le /frontend/Jenkinsfile

```
stageDevin ('🐳 Build docker', doitConstruireDocker) {
    echo "🐳 Construction Docker"
    dir('frontend') {
        dockerBuild {
            dockerContext= 'Dockerfile, dist/**, nginx.conf'
            repo = NOM_PROJET_INCA
            loginInca = 'startix-inca-app'
            imgName = NOM_DOCKER_IMAGE
            versionImg = buildVersion
            targetPtf = SLAVE_DOCKER
            push = doitLivrer
            prefixRegistry = true
            keepBuiltImage = false
            options = ['build-arg': ["VCS_REF=${gitResult.GIT_COMMIT}", "VCS_URL=${gitResult.GIT_URL}"]]
    }
}
```

L'étape BuildDocker est activée si le parametre boolean `doitConstruireDocker = true`.
Le parametre boolean `doitLivrer` sera activé lors du lancement des tâches de livraison (activer par la pipeline IC/DC)
Voici les paramètres de la methode `dockerBuild` qui permet de construire et pusher l'image vers le registry INCA:

- `dockerContext` - Fichiers nécessaire pour la construction de l'image Docker. Ce paramètre doit être sous la forme d'une expression régulière.
- `repo` - Dépôts INCA sur lequelle sera stockée l'image docker. Correspond généralement au nom du projet.
- `repoList` - Liste de dépôts supplémentaires où sera stockée l'image Docker sur INCA. Il s'agit d'une liste de noms séparée par une virgule
- `imgName` - Nom de l'image Docker construite.
- `versionImg` - Le numéro de version (tag) de l'image construite.
- `additionalTags` - Une liste de tags supplémentaires à push sur le registre (seulement si `push == true`). L'image n'aura pas ces tags localement.
- `targetPtf` - Correspond au Slave sur lequel l'image Docker est construite. Pour créer un slave Jenkins, merci de se référer à la section dédiée.
- `push` - Permet de pousser l'image Docker sur la registry INCA.
- `loginInca` - L'ID du credential INCA permettant de se connecter au registre projet.
- `prefixRegistry` - Préfixe le nom de l'image avec le nom du registre. True par défaut.
- `cleanWorkspace` - Permet de supprimer le workspace sur le Slave où est construite l'image Docker.
- `path` - Chemin vers le Dockerfile et son contexte. Généralement le Dockerfile se trouve à la racine du projet sur Gitlab.
- `registry` - Adresse du registre docker. INCA par défaut.
- `logout` - Active le logout dans le cas autre que docker.withRegistry (mot de passe avec caractères spéciaux), true par défaut .
- `keepBuiltImage` - Indique si on conserve les images construites (par défaut true).
- `options` : Ce paramètre prend la forme d'une Map dont la clef est une option de la commande docker build ("target", "build-arg", ...) et la valeur associée est une liste avec la(les) valeur(s) prise(s) par l'option. Il s'agit notamment du moyen de spécifier un proxy à utiliser lors du build, avec l'option `build-arg` pour spécifier `https_proxy`.
- `useNexus` : Paramètre booleen, si true on effectue un docker login vers le proxy-docker Nexus. Par défaut, false.
- `loginNexus` : L'ID du credential permettant le login vers le proxy-docker Nexus. Par défaut, jenkins-gitlab-devin

Administratif

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	Jenkins	Offre DevOps	https://gopro-collaboratif.rte-france.com/pages/viewpage.action?pageId=100835369

À propos

ESLint est un outil open source qui permet de détecter et corriger automatiquement les erreurs dans ton code JavaScript ou TypeScript. Il analyse ton code pour repérer :

- Les erreurs de syntaxe
- Les mauvaises pratiques
- les violations de conventions de style

Tutoriel / Intégration

1. Ajouter ESLint dans un projet TypeScript

- Ajouter les dépendances dans le fichier package.json du répertoire /frontend

```
npm install --save-dev eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin angular-eslint typesc
```

- Ajouter le script dans le fichier package.json

```
"scripts": {
  "lint": "eslint ."
}
```

2. Ajouter la configuration

- Ajouter le fichier de configuration eslint.config.mjs à la racine du projet.
- Voici le contenu du fichier :

```
import eslint from '@eslint/js';
import angular from 'angular-eslint';
import globals from 'globals';
import typescript from 'typescript-eslint';

export default typescript.config(
  {
    languageOptions: {
      globals: {
        ...globals.node,
      },
    },
  },
  {
    ignores: ['dist/', 'target/', '.angular/'],
  },
  eslint.configs.recommended,
  {
    files: ['src/**/*.ts'],
    extends: [...typescript.configs.strictTypeChecked, ...typescript.configs.stylistic, ...angular.configs.tsR
    languageOptions: {
      globals: {
        ...globals.browser,
      },
      parserOptions: {
        project: ['./tsconfig.app.json', './tsconfig.spec.json'],
      },
    },
    processor: angular.processInlineTemplates,
    rules: {
```

```
      '@angular-eslint/component-class-suffix': 'off',
      '@angular-eslint/component-selector': [
        'error',
        {
          type: 'element',
          style: 'kebab-case',
        },
      ],
      '@angular-eslint/directive-selector': [
        'error',
        {
          type: 'attribute',
          style: 'camelCase',
        },
      ],
      '@typescript-eslint/unbound-method': 'off',
      '@typescript-eslint/no-floating-promises': 'off',
      '@typescript-eslint/no-extraneous-class': 'off',
      '@typescript-eslint/no-confusing-void-expression': 'off',
      '@typescript-eslint/no-unsafe-assignment': 'off',
      '@typescript-eslint/no-explicit-any': 'off',
      '@typescript-eslint/no-non-null-assertion': 'off',
      '@typescript-eslint/no-unsafe-member-access': 'off',
      '@typescript-eslint/no-unsafe-argument': 'off',
      '@typescript-eslint/restrict-template-expressions': ['error', { allowNumber: true }],
      'arrow-body-style': 'error',
    },
  },
  {
    files: ['**/*.html'],
    extends: [...angular.configs.templateRecommended, ...angular.configs.templateAccessibility],
  },
);
```

3. Lancer l'analyse eslint

- Dans le dossier frontend, executer :

```
npm run lint
```

- Et pour corriger automatiquement :

```
npm run lint -- --fix
```

À propos

Kubernetes, souvent abrégé en K8s, est une plateforme conçue pour automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Développé par Google et maintenant maintenu par la Cloud Native Computing Foundation (CNCF), Kubernetes permet aux équipes de développement et d'exploitation de gérer des clusters de conteneurs de manière efficace et fiable.

Contexte

Kubernetes (K8s) est une plateforme open source qui automatise le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Il est devenu un standard de facto dans l'industrie pour plusieurs raisons :

- Standardisation des environnements : facilite la cohérence entre développement, test et production.
- Scalabilité automatique : ajuste les ressources en fonction de la charge.
- Haute disponibilité : redémarre automatiquement les services en cas de défaillance.
- Portabilité : compatible avec les principaux fournisseurs cloud et les infrastructures on-premise.
- Optimisation des coûts : meilleure utilisation des ressources matérielles.

✅ Avantages fonctionnels : - Agilité accrue : accélère les cycles de développement et de mise en production. - Réduction des risques : grâce à l'isolation des services et à la gestion automatisée des défaillances. - Flexibilité organisationnelle : facilite la collaboration entre équipes Dev, Ops et métier. - Visibilité et contrôle : permet un suivi précis des performances et des ressources.

Tutoriel / Intégration

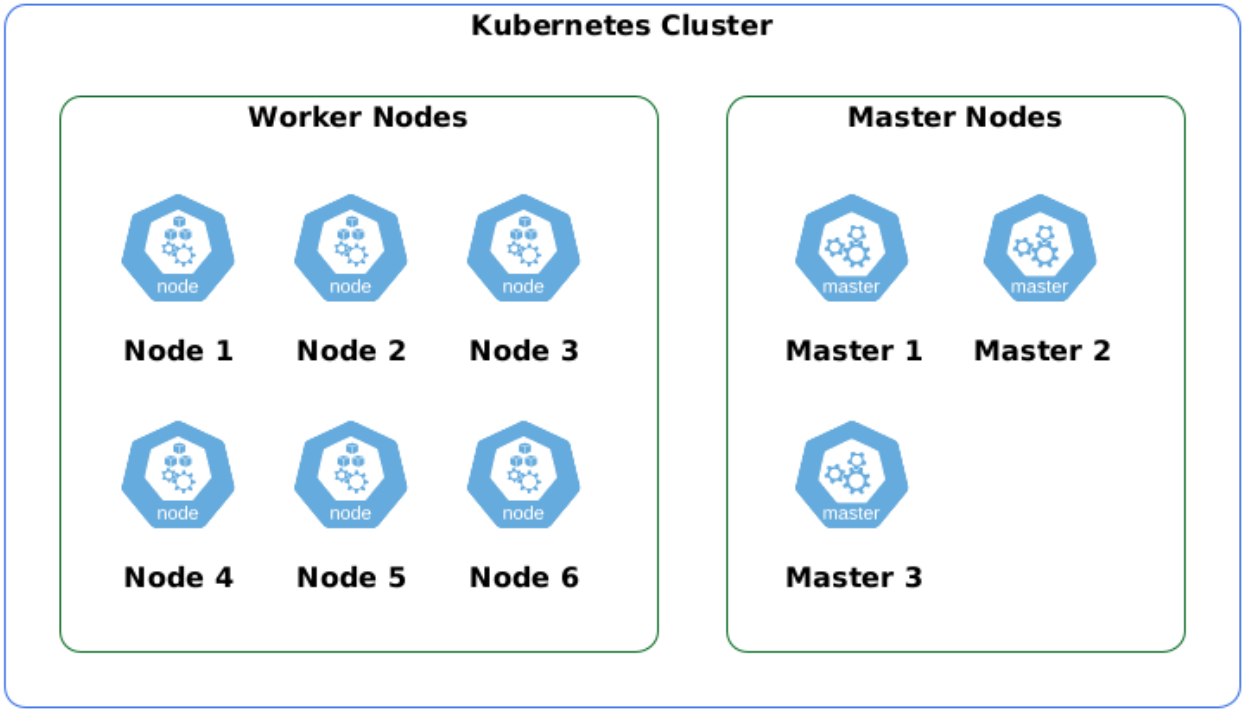
L'accès à un cluster Kubernetes peut se faire de différentes manières, en fonction des besoins spécifiques de votre environnement. Ce chapitre couvre les prérequis, ainsi que plusieurs options pour disposer d'un cluster Kubernetes.

Architecture

L'architecture de Kubernetes est composée de plusieurs composants clés qui interagissent pour gérer le déploiement et la maintenance des applications conteneurisées.

Nodes

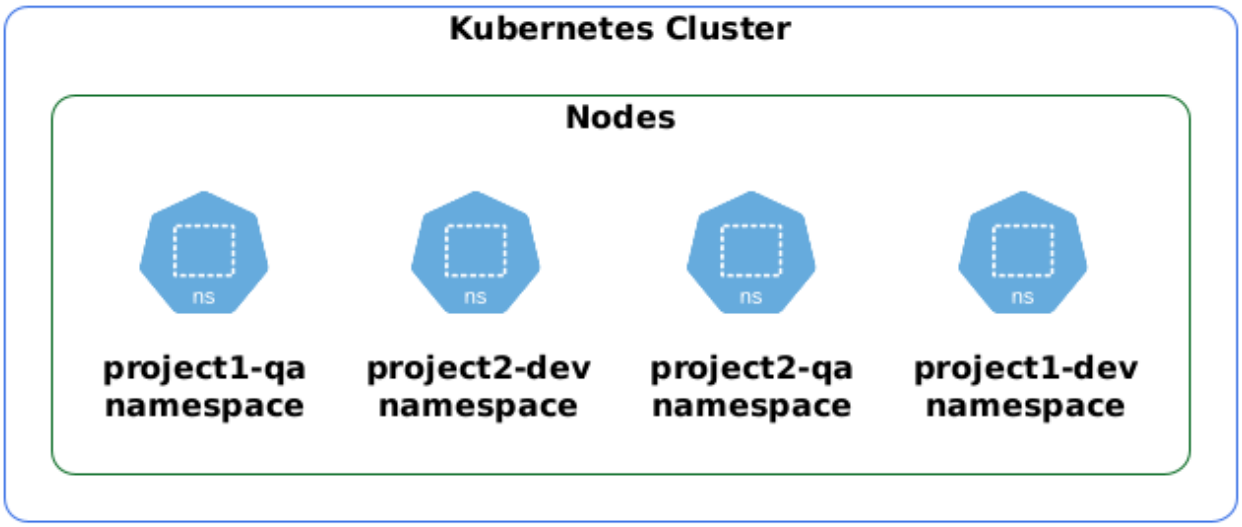
Un node est une machine physique ou virtuelle sur laquelle Kubernetes est installé.



Kubernetes nodes

Namespace

Un Namespace en Kubernetes est un moyen de diviser les ressources d'un cluster en plusieurs environnements virtuels. Les Namespaces sont principalement utilisés dans les environnements où plusieurs équipes ou projets partagent le même cluster, mais doivent être isolés les uns des autres.



Kubernetes namespaces

Pods

Un pod est l'unité de base de Kubernetes. Il représente un ou plusieurs conteneurs qui partagent le même réseau et le même espace de stockage. Les pods sont éphémères et peuvent être recréés en cas de défaillance.

Services

Les services sont des abstractions qui définissent un ensemble logique de pods et une politique d'accès pour les atteindre. Ils permettent une découverte et un équilibrage de charge faciles entre les pods.

Ingress

Un Ingress en Kubernetes est une ressource qui gère l'accès externe aux services dans un cluster, généralement via HTTP et HTTPS. Ingress fournit des règles pour diriger le trafic vers les services Kubernetes en fonction de l'URL de la demande, des en-têtes HTTP, etc

Volumes

Les volumes sont utilisés pour stocker des données de manière persistante. Contrairement aux données stockées dans des conteneurs, les volumes persistent même si le pod est recréé.

ConfigMap

Un ConfigMap est une ressource Kubernetes utilisée pour stocker des données de configuration sous forme de paires clé-valeur. Elle permet de séparer la configuration d'une application de son code, facilitant ainsi la gestion des environnements (dev, staging, prod).

Secret

Un Secret est une ressource Kubernetes qui stocke de manière sécurisée des données sensibles comme des mots de passe, des jetons ou des clés, afin d'éviter de les exposer directement dans les fichiers de configuration ou les images de conteneur.

Structure de la configuration K8S

Pour intégrer le déploiement Kubernetes, il faut ajouter cette structure :

Par exemple: pour le projet Startix, la structure k8s existe sous le dossier `deployment` à la racine du projet.

```
📁 k8s
├── 📁 components : les composants kubernetes
│   ├── 📁 startix-back : nom du composant backend
│   │   ├── 📄 configmap.yaml
│   │   ├── 📄 deployment.yaml
│   │   ├── 📄 ingress.yaml
│   │   ├── 📄 kustomization.yaml │ │ ├── 📄 cacerts-secret.yaml
│   │   └── 📄 service.yaml
│   ├── 📁 startix-front : nom du composant frontend
│   │   ├── 📄 configmap.yaml
│   │   ├── 📄 deployment.yaml
│   │   ├── 📄 kustomization.yaml
│   │   ├── 📄 ingress.yaml
│   │   └── 📄 service.yaml
├── 📁 ns-startix-dev : le nom du namespace Startix de l'env dev
│   ├── 📄 kustomization.yaml
│   ├── 📄 patch_configmap_frontend.yaml
│   └── 📄 patch_ingress.yaml
├── 📁 ns-startix-int : le nom du namespace Startix de l'env int
│   ├── 📄 kustomization.yaml
│   ├── 📄 patch_configmap_frontend.yaml
│   └── 📄 patch_ingress.yaml
├── 📁 ns-startix-prd : le nom du namespace Startix de l'env prod
│   ├── 📄 kustomization.yaml
│   ├── 📄 patch_configmap_frontend.yaml
│   └── 📄 patch_ingress.yaml
...
```

kustomization.yaml

kustomize est un outil de gestion des configurations Kubernetes qui permet de personnaliser des manifestes YAML sans avoir besoin de les modifier directement. Il utilise une approche déclarative pour appliquer des patchs et des transformations aux ressources Kubernetes.

kustomize est intégré à kubectl, donc aucune installation séparée n'est nécessaire.

Pour l'installation de `kubectl`, merci de voir la section dédiée.

```
// exemple d'une commande kubectl
kubectl apply -k ./dossier-kustomize
```

Dans le projet startix, le fichier ns-startix-dev/kustomization.yaml est le point d'entrée principal pour déployer les ressources kubernetes sur l'environnement `dev`. Cela permet de modulariser le déploiement par composant (par exemple, avoir un composant pour la base de données, un autre pour l'authentification, etc) et par environnement (dev, recette, preprod, prod, etc).

Sans dupliquer les fichiers YAML, Kustomize permet de gérer des configurations Kubernetes pour plusieurs environnements (comme dev, recette, prod). Pour ajouter un déploiement pour un nouveau environnement il faut: - Créer un nouveau dossier avec le nom du namespace (exemple: `ns-startix-prd`) sous le dossier k8s (voir la structure ci-dessus). - Ajouter le fichier kustomization.yaml en adaptant les valeurs (namespace, image newName, image newTag) - Ajouter le fichier patch_ingress.yaml en adaptant la valeur du host. - Ajouter le fichier patch_configmap_frontend.yaml en adaptant les valeurs des variables d'environnement du composant Frontend. - Garder la même configuration dans les composants back et front sous `/composnents` et les fichiers standards de déploiement (deployment, service, ingress, etc.)

Un composant est une unité réutilisable sur différents environnements qui peut contenir des ressources Kubernetes, des patches, des configurations, etc.

- composants:

```
components:
  - ../components/startix-back
  - ../components/startix-front
```

Cette section inclut un Kustomize situé dans les composants startix-back et startix-front.

- patches:

```
patches:
  - path: patch_ingress.yaml
    target:
      kind: Ingress
```

Cette section applique un patch (modification) à une ressource Kubernetes de type `Ingress`. Le fichier ns-startix-dev/patch_ingress.yaml contient la modification à apporter sur la ressource ingress.

```
  - path: patch_configmap_frontend.yaml
    target:
      kind: ConfigMap
      name: startix-front-public-env
```

Cette section applique une modification à une ressource Kubernetes de type `ConfigMap` avec le nom `startix-front-public-env`. Le fichier ns-startix-dev/patch_configmap_frontend.yaml contient la modification à apporter sur la ressource ConfigMap.

patch_ingress.yaml

Ce patch modifie une ressource Ingress.

```
- op: replace
  path: /spec/rules/0/host
  value: startix-dev.rte-france.com
```

Cette opération permet de remplacer la valeur du champ `host` dans les ressources ingress par le domaine de l'environnement dev du startix, ce qui permet d'exposer l'application sur `startix-dev.rte-france.com`.

patch_configmap_frontend.yaml

Ce patch modifie la ressource ConfigMap du composant frontend.

```
- op: replace
  path: /data/OIDC_REDIRECT_URI
  value: https://startix-dev.rte-france.com/startix-front/
```

Cette opération permet de remplacer la valeur de la variable d'environnement `OIDC_REDIRECT_URI` dans la configMap Frontend par la valeur correspondante à l'environnement dev du startix.

ingress.yaml

L'ingress permet d'exposer notre Service à l'extérieur d'un cluster, sur un « ingress-controller ».

Exemple du fichier ingress.yaml du composant startix-back:

```
---
kind: Ingress
apiVersion: networking.k8s.io/v1
metadata:
  annotations:
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
  name: startix-back-ingress
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
        - path: /startix-back(/|$)(.*)
          pathType: ImplementationSpecific
          backend:
            service:
              name: startix-back
              port:
                number: 80
```

Beaucoup de personnalisation sont possibles, notamment pour exposer différents « path » ou ajouter des annotations pour modifier certaines options côté Loadbalancer, comme des règles de Rewrite, ou autres options de taille de body...

annotations: directives spécifiques à NGINX pour affiner le comportement de l'Ingress. - **use-regex**: "true" : autorise l'usage de regex dans le champ path (très utile pour des règles flexibles). - **rewrite-target**: permet de réécrire l'URL d'entrée en retirant /startix-back, pour rediriger vers / dans le service (en utilisant la capture group $2).

rules: définit la logique de routage pour les requêtes entrantes. - **path**: /startix-back(/|$)(.): toutes les requêtes commençant par /startix-back (suivies de / ou rien) sont capturées. Grâce à use-regex, cette expression est interprétée comme une regex. - pathType*: ImplementationSpecific: laisse le contrôleur choisir comment interpréter le chemin (regex, préfixe, etc.).

backend: définit le service cible interne - **name**: startix-back: nom du service Kubernetes à contacter - **port**: 80: port exposé du service

Ce que fait l'ingress en résumé - Il capte les requêtes qui ciblent /startix-back/.... - Réécrit les URLs pour les transmettre proprement au service (ex : /startix-back/api devient /api côté backend). - Ajoute les headers nécessaires pour transmettre les infos de client (IP, host). - Utilise NGINX comme contrôleur pour gérer le routage.

service.yml

Pour créer un Service, il faut lui indiquer grâce à un « selector » comment il doit sélectionner ses Pods. Ce selector s'appuie sur les labels que vous configurez dans votre Deployment. C'est pour cela que les labels sont importants.

Exemple du fichier service.yaml du composant startix-back:

```
---
apiVersion: v1
kind: Service
metadata:
  name: startix-back
  labels:
    app: startix-back
spec:
  selector:
    app: startix-back
  ports:
    - name: web
      protocol: TCP
      port: 80
      targetPort: 8080
```

Un service peut exposer un ou plusieurs ports : - **targetPort** correspond au port du Container défini dans le deployment.yaml. - **port** est le port exposé par le Service.

deployment.yaml

La configuration d'un déploiement sur Kubernetes se faire à l'aide d'un deployment.yaml.

Voici une explication concise pour le mapping des variables d'environnement et des volumes :

- **Variables d'environnement** :
  - `configMapRef` : référence le **ConfigMap** contenant les variables d'environnement publiques.
  - `secretRef` : référence le **Secret** contenant les variables d'environnement sensibles ou privées.

- **volumeMounts** :
  - Monte le fichier cacerts dans le conteneur à l'emplacement `/etc/ssl/certs/java/cacerts`, utilisé pour les certificats SSL Java à partir de l'option `javax.net.ssl.trustStore` dans l'image Docker. (Voir Dockerfile).
- **volumes** :
  - Source : `cacerts-secret`, le nom du Secret contenant le fichier de certificat.
  - Contenu : la clé `cacerts` est extraite et montée dans le conteneur via le volume.

Remarque: le Secret de certificat cacerts-secret.yaml est crée via cette commande `kubectl` :

```
kubectl create secret generic cacerts-secret --from-file=cacerts={PATH_TO_CACERTS_FILE} --dry-run=client -o ya
```

Merci de consulter l'exemple détaillé d'un deploiement simple avec les bonnes pratiques sur cette page.

L'exposition du déploiement repose sur les deux mécanismes essentiels :

- **Le Service**, qui représente un équilibreur de charge interne qui pointe sur un ou plusieurs Pods en statut « Ready »
- **L'Ingress**, qui expose sur un équilibreur de charge externe une règle permettant l'accès depuis l'extérieur à notre Service

configmap.yaml

Le fichier configmap.yaml contient les variables d'environnement des composants : - Backend : startix-back/configmap.yaml - Frontend : startix-front/configmap.yaml

Voici un exemple :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: startix-front-public-env
data:
  OIDC_ISSUER: "https://gaia-sso.opf.rte-france.com"
  OIDC_CLIENT_ID: "startix-dev-front-opf"
  OIDC_REDIRECT_URI: "http://localhost:9000/"
```

- **name**: Le nom de la ressource ConfigMap.
- **data**: La définition des variables de configuration de l'application.

secret.yaml

Le fichier secret.yaml contient les données sensibles comme des mots de passe, des jetons ou des clés. C'est une ressource kubernetes de type `Opaque` qui contient les données sensibles chiffrées en `base64`.

Voici un exemple :

```
apiVersion: v1
kind: Secret
metadata:
  name: startix-secret
type: Opaque
data:
  SPRING_DATASOURCE_USERNAME: "cGd1XzAxbzEwNDdfbw=="
  SPRING_DATASOURCE_PASSWORD: "YmFwMUJZWHQ4Mg=="
```

- Le secret est généré par le job **JenkinsDeploy**, qui récupère automatiquement les données sensibles depuis **Vault**. Voir la documetation Vault.md pour plus de détails.

Prérequis

Demande de namespace ik8s

La documentation de l'offre Kubernetes RTE IK8S est ici : IK8S Home

Une demande de namespace dans un cluster K8S est à faire à l'équipe DESIGN.

Voici la demande pour le namespace "startix-dev" nécessaire à l'environnement de développement.

```
Bonjour,

Nous aurions besoin d'un namespace K8S pour le projet Startix:

    NNA de Startix : 1047
    nom du namespace : ns-startix-dev
```

```
Pour le capacity planning, ils nous faudrait des ressources pour une dizaine de pods.
Une ébauche d'architecture se trouve ici : https://gopro-collaboratif.rte-france.com/display/SAX/Architecture

Nous demanderons plus de ressources par la suite si nécessaire.

Il nous faudrait également un jenkins slave dans ce namespace.

Nous avons déjà :

    un projet Gilab : https://devin-source.rte-france.com/groups/startix
    un projet Jenkins : https://devin-ic.rte-france.com/job/startix/
    un projet Inca : https://inca.rte-france.com/harbor/projects/143/summary

L'équipe Startix
```

Accès au cluster Kubernetes k8s avec Kubectl

kubectl est l'outil en ligne de commande principal pour interagir avec des clusters Kubernetes. Il est utilisé pour déployer et gérer vos applications.

Installer kubectl si ce n'est pas déjà fait

```
curl -LO https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl &&
```

Vous devez ensuite récupérer le fichier kubeconfig.yaml : - aller à l'URL https://login-ik8s-opf.rte-france.com et cliquer sur le lien "Full Kubeconfig" à gauche - copier le contenu de "Full Kubeconfig" dans le fichier ~/kubeconfig.yaml - modifier l'attribut namespace par la valeur de votre namespace, par exemple pour le namespace ns-startix-dev : - namespace: ns-startix-dev

Note: Refaire la procédure précédente lorsque le token expire

Il faut ensuite exposer la variable **KUBECONFIG** avec pour valeur le chemin vers le fichier `kubeconfig.yaml`, par exemple :

```
export KUBECONFIG={ABSOLUTE-PATH-TO}/kubeconfig.yaml
```

Il est également possible de persister cette configuration dans la configuration de votre shell. Voici un exemple pour le shell bash.

```
# Ajout de la config K8s dans ~/.bashrc
cat <<EOF >>  ~/.bashrc
## K8s
# Ajouter l'IP 10.132.148.117 dans la variable NO_PROXY pour K8s
export NO_PROXY=${NO_PROXY},10.132.148.117

# Donner le chemin kubeconfig à kubectl
export KUBECONFIG=~/kubeconfig.yaml

# Ajouter le chemin du binaire kubectl au PATH
export PATH=$PATH:$(dirname `which kubectl`)

# Ajouter la completion pour bash
source <(kubectl completion bash)
EOF
```

Voici quelques commandes de base pour vous familiariser avec cet outil:

```
# Afficher les contextes disponibles
kubectl config get-contexts

# Définir le contexte actuel
kubectl config use-context my-cluster

# Afficher les noeuds du cluster
kubectl get nodes

# Afficher les pods dans tous les namespaces
kubectl get pods --all-namespaces

# Créer un namespace
kubectl create namespace my-namespace

# Appliquer une configuration à partir d'un fichier YAML
kubectl apply -f my-config.yaml
```

```
# Créer un déploiement nginx
kubectl create deployment nginx --image=nginx

# Exposer le déploiement via un service
kubectl expose deployment nginx --type=NodePort --port=80

# Obtenir les informations du service
kubectl get svc nginx
```

Voici la commande kubectl qui permet le deploiement du projet Startix.

```
export NAMESPACE=ns-startix-dev
kubectl apply --namespace ${NAMESPACE} -k deployment/k8s/${NAMESPACE}
```

Voici la commande kubectl qui permet de supprimer le deploiement du projet Startix.

```
export NAMESPACE=ns-startix-dev
kubectl delete --force --grace-period=0 --namespace ${NAMESPACE} -k deployment/k8s/${NAMESPACE} || true
```

Automatisation du deploiement k8s avec Jenkinsfile

Pipelines

Pour pouvoir exécuter le job devin-k8s-link/main sur devin-dc.rte-france.com, nous aurions besoin que le plugin Jenkins K8S soit installé comme sur devin-ic.rte-france.com pour avoir des agents dynamiques pour la pré-prod et la prod.

De plus, il faudrait différents labels (sur devin-ic.rte-france.com et devin-dc.rte-france.com) pour déployer sur les différents clusters IK8S :

- **ik8s-opf**
- host : master-ik8s-opf.rte-france.com
- namespace : ns-devin-opf
- **ik8s-opfi**
- host: api-ik8s-opfi.rte-france.com
- namespace: ns-devin-opfi
- **ik8s-ppr**
- host: api-ik8s-preprod.rte-france.com
- namespace: ns-devin-ppr
- **ik8s-prd**
- host: api-ik8s.rte-france.com
- namespace: ns-devin-prd

Le projet Startix est deployé sur l'environnement **OPF**. Le pipeline s'exécute sur un nœud Jenkins mutualisé (node('ik8s-opf')), et suit ces étapes :

🦊 **Checkout**

- Récupère le code depuis Git et détermine la branche active.

🏗️ **Stage - Make diff**

- Compare les fichiers de déploiement Kubernetes avec l'état actuel du cluster distant (kubectl diff).

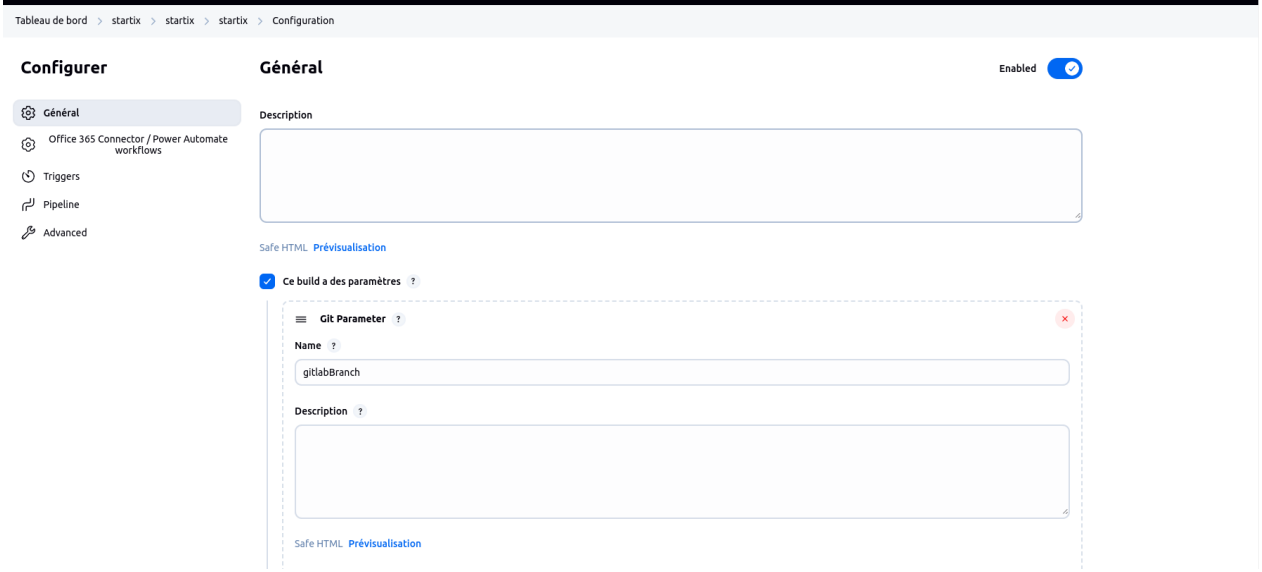
🏗️ **Stage - Deployment**

- Déploie les fichiers de déploiement Kubernetes dans le cluster kube (kubectl apply).

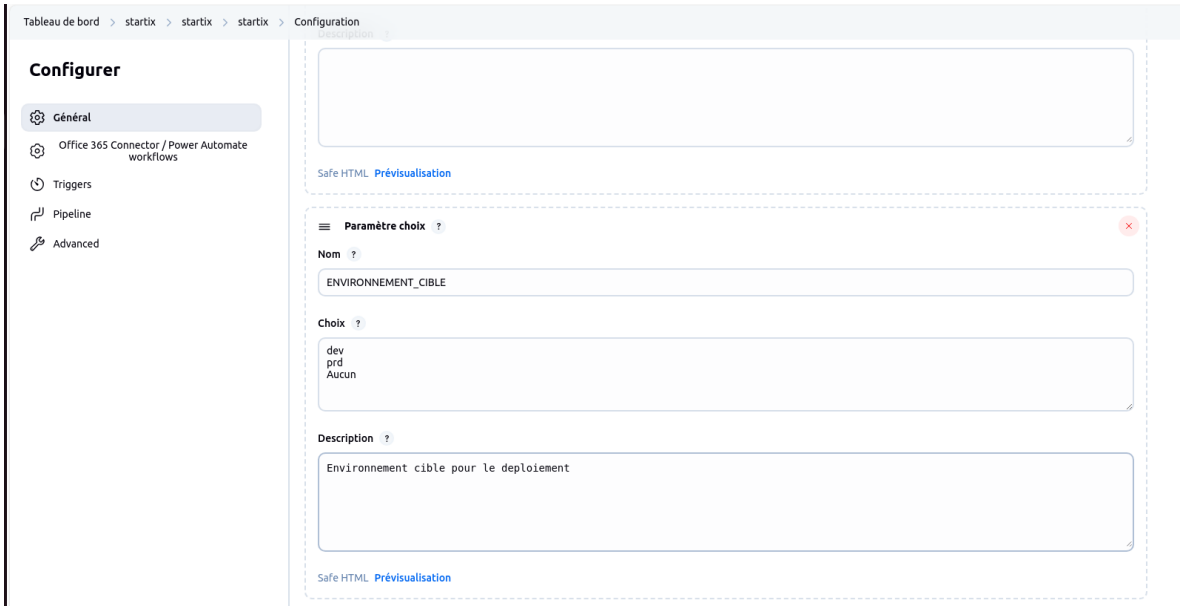
Déclaration des constantes et paramètres dans le Jenkinsfile

Pour automatiser le deploiement des ressources sur un cluster Kubernetes, il faut: - Adapter le fonctionnement du pipeline Jenkins dans le Jenkinsfile_Deploy. - Créer une nouvelle pipeline sur Jenkins pour la partie Déploiement Contenu (DC) et l'attacher au fichier `Jenkinsfile_Deploy`. (voir Comment écrire et configurer une pipeline Jenkins dans la section dédiée) - Ajouter le paramètre `ENVIRONNEMENT_CIBLE` dans la configuration de la nouvelle pipeline Jenkins.

Voici les étapes à suivre: - Aller sur la configuration du job jenkins du projet



- Ajouter un paramètre de type `paramètre choix` avec le nom `ENVIRONNEMENT_CIBLE` et mettre les environnements dans les choix.



voici un exemple du code dans le jenkinsfile :

- Le pipeline s'exécute sur un nœud Jenkins nommé **ik8s-opf** qui contient un `kubectl`.

```
// Execution sur le noeud de ik8s-opf de la plateforme Devin
node('ik8s-opf') {
  try {
    // Étapes principales
  } catch (e) {
    // Gestion des erreurs
  } finally {
    // Nettoyage du workspace
  }
}
```

- Paremètres du JOB :
  - ENVIRONNEMENT_CIBLE : l'environnement cible à deployer (dev, recette, prod, etc.). Valeur par défaut : 'Aucun'
- Utiliser la variable d'environnement env.NAMESPACE qui est composée par :
  - NOM_PROJET : nom du projet
  - environnementCible : dev | preprod | prod | Aucun ...

```
// par expemple : ns-startix-dev
env.NAMESPACE = "ns-${NOM_PROJET}-${environnementCible}"
```

- **Étape Make diff** : Affiche les différences entre les fichiers de déploiement locaux et les ressources existantes dans le cluster Kubernetes, sans bloquer le pipeline en cas d'erreur (|| true).

```
stageDevin("Stage - Make diff") {
  withKubeConfig([credentialsId: "jenkins-ic-startix-${NAMESPACE}"]) {
    sh '''
      kubectl diff --namespace ${NAMESPACE} -k deployment/k8s/${NAMESPACE} || true
      kubectl get pods --namespace ${NAMESPACE}
    '''
  }
}
```

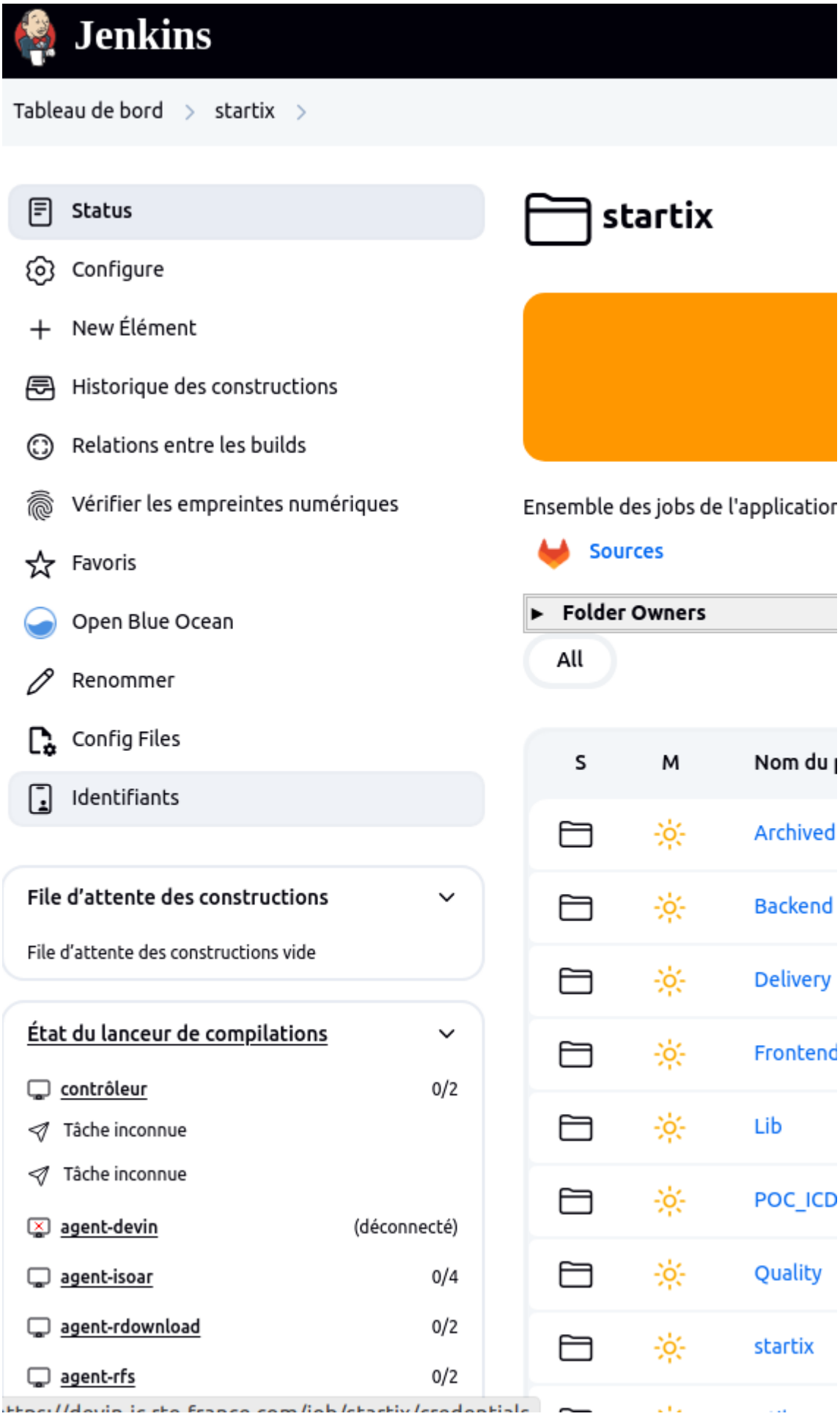
- **Étape Deployment** : Applique les configurations Kubernetes (apply -k) et liste les pods après déploiement.

```
stageDevin("Stage - Deployment") {
    withKubeConfig([credentialsId: "jenkins-ic-startix-${NAMESPACE}"]) {
      sh '''
        kubectl apply --namespace ${NAMESPACE} -k deployment/k8s/${NAMESPACE}
        kubectl get pods
      '''
    }
}
```

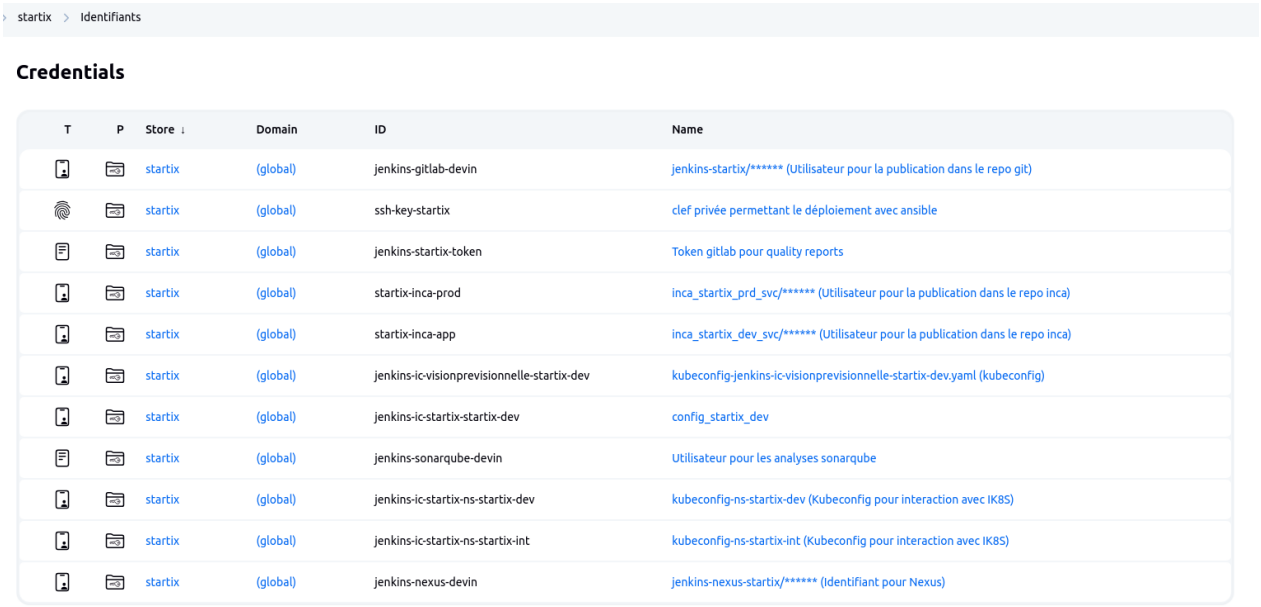
NOTE: Utiliser le `credentialsId` lié au namespace du projet dans la methode `withKubeConfig`.

Le **credentialId** doit être généré automatiquement lors de la création du namespace. Ce credential doit être recherché dans les identifiants Jenkins associés au projet. S'il n'est pas présent, il convient de contacter l'équipe Devin pour le fournir, ou créer une demande de support sur le DEVMCO.

- Ouvrer la page jenkins du projet et selectionner dans le menu à gauche "identifiants"


Jenkins
Tableau de bord
startix
Status
Configure
New Élément
Historique des constructions
Relations entre les builds
Vérifier les empreintes numériques
Favoris
Open Blue Ocean
Renommer
Config Files
Identifiants
File d'attente des constructions
File d'attente des constructions vide
État du lanceur de compilations
contrôleur 0/2
Tâche inconnue
Tâche inconnue
agent-devin (déconnecté)
agent-isoar 0/4
agent-rdownload 0/2
agent-rfs 0/2
startix
Ensemble des jobs de l'applicatior
Sources
Folder Owners
All
S
M
Nom du
Archived
Backend
Delivery
Frontend
Lib
POC_ICD
Quality
startix

• Utiliser le credential ID associé à la configuration kubernetes



Lancement manuel d'un job Jenkins

• Lancer un build de déploiement avec des paramètres sur l'interface Jenkins.
• Choisir l'environnement cible à deployer (DEV | INT | PROD...) puis lancer le JOB.

Administratif

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	Ik8s	Infrastructure	- https://gopro-collaboratif.rte-france.com/display/IK8S/IK8S+Home - https://gopro-collaboratif.rte-france.com/pages/viewpage.action?pageId=431404385

Documentation Marvel

À propos

Marvel (Monitoring Analyse Recherche Visualisation avec ELk) permet de centraliser les logs d'une application, ce qui est utile quand l'application a une architecture micro-services ou est décomponsée en plusieurs containers.

En effet, la centralisation des logs permet, comme son nom l'indique, d'avoir tous les logs d'une application au même endroit, et donc d'avoir un meilleur suivi de ce qui se passe, notamment lorsque le back et le front interagissent par exemple.

Offre de service

Marvel est une offre de service. À ce titre, il existe une page de documentation rapide ainsi qu'un lien pour créer une demande Cosmos. Vous pourrez trouver toutes les informations utiles ici.

Il est également possible de plus de détails sur cette page.

Il existe également des pages Gopro sur MARVEL sur l'espace DevFactory et sur l'espace OSA.

Toutes vos demandes ou toutes questions seront à adresser à la liste DSIT--DESIGN-OSA-SUPERVISION

Tutoriel / Intégration

Marvel est disponible sur les environnements : - **OPF** - En **préproduction** et en **production**

Marvel peut s'intégre en 2 étapes : 1. Récupération des logs en local (avec **Docker** ou **docker-compose**) 2. Envoi des logs à Marvel pour les applications distribuées utilisant - **Kubernetes** à venir prochainement - **Ansible** en suivant cette documentation (non mis en place par Startix)

Le projet Startix centralise les logs en local avec **docker-compose** : - Les containers définis dans le docker-compose.yml montent des volumes partagés entre l'hôte et l'application avec les droits d'écriture - Les logs sont écrits directement dans les fichiers présents dans les volumes partagés.

Récupération des logs en local

Introduction

Dans cette partie, on cherche à récupérer tous les logs de tous les micros-services de notre application sur la machine hôte. Startix a redirigé ses logs dans des fichiers distincts pour chaque micro-service : * un fichier au format `.log` qui contient les logs bruts * un ficher au format `.json` qui contient les logs enrichis au format JSON. Les logs enrichis contiennent des métadonnées supplémentaires, comme l'heure à laquelle le log a été écrit, l'adresse IP et le nom de la machine sur laquelle le micro-service est lancé, l'environnement utilisé (dev, prod, ... créé au moment du lancement de l'application), ...

Pré-requis

Installation du logger Elan de RTE

Pour installer elan, il faut rajouter ce bloc dans le pom.xml du projet :

```
    <dependency>
        <groupId>com.rte_france.elan</groupId>
        <artifactId>elan-logger</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </dependency>
```

Remarque : En local, faites attention à bien ajouter ce profile dans votre fichier ~/.m2/settings.xml :

```
<profile>
    <id>devin-nexus-snapshots</id>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
    <!-- Repository permettant de telecharger des SNAPSHOTS -->
        <repository>
            <id>nexus</id>
            <name>nexus</name>
            <url>https://devin-depot.rte-france.com/repository/maven-snapshots/</url>
        </repository>
    </repositories>
</profile>
```

Si vous n'avez pas ce profile, vous n'arriverez pas à télécharger le snapshot de elan-logger. Cette dépendance fournit un provider au sens logback pour construire chaque évènement de log et l'enrichir avec les informations utiles à inclure dans les logs résultants.

logback-spring.xml

Ce fichier sert, pour chaque microservice, à paramétrer le fonctionnement des logs (format, variables globales ...):

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/defaults.xml"/>
    <!-- Spring Log properties -->
    <springProperty scope="context" name="springApplicationName" source="spring.application.name"/>
    <springProperty scope="context" name="springActiveProfile" source="spring.profiles.active"/>
    <!-- Log properties -->
    <property scope="context" name="imageName" value="${APPLICATION_NAME}_back"/>
    <property scope="context" name="containerName" value="${APPLICATION_NAME}_back"/>
    <property scope="context" name="vmIP" value="${HOST_IP}"/>
    <property scope="context" name="vmName" value="${HOST_NAME}"/>

    <!-- Classic appender to log console-->
    <appender name="Console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level %class{36} %M %L - %msg%n</pattern>
        </encoder>
    </appender>

    <!-- Classic appender to log file-->
    <appender name="RollingFile" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>./logs/application.log</file>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <charset>UTF-8</charset>
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level %M %class{0} %L - %msg%n</pattern>
        </encoder>
```

```
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- Rollover daily and when the file reaches 10 MegaBytes -->
            <fileNamePattern>/logs/application-%d{yyyy-MM-dd}.log.gz</fileNamePattern>
            <maxHistory>150</maxHistory>
        </rollingPolicy>
    </appender>
    <!-- Appender to log to file in a JSON format, one JSON object per line -->
    <appender name="JSON"
              class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>./logs/${APPLICATION_NAME}_back_application.json</file>
        <rollingPolicy
                class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
            <fileNamePattern>./logs/${APPLICATION_NAME}_back_application.%d{yyyy-MM-dd}.%i.json</fileNamePatte
            <maxFileSize>10MB</maxFileSize>
            <maxHistory>3</maxHistory>
            <totalSizeCap>20MB</totalSizeCap>
        </rollingPolicy>
        <encoder
                class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <provider
                        class="com.rte_france.elan.logger.provider.LogDataJsonProvider">
                </provider>
            </providers>
        </encoder>
    </appender>

    <!-- LOG everything at INFO level -->
    <root level="info">
        <appender-ref ref="RollingFile" />
        <!-- Comment this appender to disable console logging in the Prod environment -->
        <appender-ref ref="Console" />
        <appender-ref ref="JSON"/>
    </root>
</configuration>
```

Ce fichier sera par la suite copié dans le container via la ligne suivante du Dockerfile :

```
COPY src/main/resources/logback-spring.xml /app/logback-spring.xml
```

Ainsi, l'application lancée depuis le container saura comment formater les logs, même si seul le `.jar` du projet est présent sur le container (et non le code source du micro-service).

Récupération des logs

Le fichier docker-compose.yml lance tous les containers nécessaires au fonctionnement de l'application et monte les volumes nécessaires pour récupérer les logs au format json et .log chez la machine hôte.

> **Remarque:** Même s'il est possible de lancer le project directement en utilisant la commande `docker-compose`, lors du premier lancement, il est nécessaire d'utiliser le script docker.sh pour créer les dossiers et fichiers de logs avec les bonnes permissions (pour permettre à l'hôte et au container de lire et écrire les logs sans passer par les droits root).
>
> Pour ce faire, lancer la commande
>
> `bash ./docker.sh --app MyApp ./docker.sh --app MyApp --build # Alternative pour builder les images Docker`
>
> Pour plus d'informations sur le script docker.sh, lancer
>
> `bash ./docker.sh --help`
>
> Le fichier docker.sh est aussi utilisé pour indiquer les données de la machine hôte (adresse IP et nom). De cette façon, les logs enrichis par le logger elan de RTE a plus de données à fournir (cf ci-dessous).

Dans la configuration proposée par Startix, les logs que l'application génrère sont dans les dossiers: * `./logs/` pour les logs au format `json`. On y trouvera les fichiers \${APPLICATION_NAME}_backend.json et \${APPLICATION_NAME}_frontend.json * `./config/logs/` pour les logs au format texte * `backend/logs/` qui contient deux fichiers : `application.log` qui contient les logs bruts, et le fichier `${APPLICATION_NAME}_back_application.json` qui contient les logs au format json enrichis par le logger elan de RTE. Elan ajoute automatiquement certaines informations de contexte dans les logs, comme la date, l'heure, l'environnement et l'hôte dans lequel est lancée l'application. * de façon similaire, le dossier `frontend/logs` contient les logs du front, au format texte et json

Exemple de logs aux différents formats

Pour plus de concret, voici un exemple de log au format brut:

```
2025-12-19T16:15:16.334+01:00 WARN  main StartixApplication 18 - This is a warning
```

Et voici son équivalent au format JSON


```
{"timestamp":"2025-12-19T15:15:16.334Z",
  "labels":{"env":"demo","app":"startix"},
  "ecs":{"version":"1.0.0"},
  "host":{"hostname":"gm0winl896.bureau.si.interne","ip":"127.0.0.1"},
  "container":{
    "id":"Image not found",
    "name":"fb7b8f5f0ffb",
    "ip":"172.23.0.5",
    "type":"startix_back",
    "image":{"name":"startix_back"}
  },
  "event":{
    "id":"c5c87e41-5fe8-416b-97e9-e65036528465",
    "start":"2025-12-19T16:15:16.334Z",
    "end":"2025-12-19T16:15:16.334Z",
    "action":"main",
    "dataset":"com.rte.fr.startix.StartixApplication",
    "original":"This is a warning"
  },
  "log":{
    "level":"WARN",
    "origin":{
      "function":"main",
      "file":{"name":"StartixApplication.java","line":18}}
  }
}
```


Il s'agit d'un log tout simple ajouté avec

```
import org.slf4j.LoggerFactory;
import org.slf4j.Logger;

@SpringBootApplication
public class StartixApplication {

    private static final Logger LOGGER = LoggerFactory.getLogger(StartixApplication.class);

    public static void main(String[] args) {
        ...
        LOGGER.warn("This is a warning");
    }

}
```

On voit ici clairement que le format json produit par le logger elan a beaucoup enrichi le log, ce qui peut être utile selon le traitement qu'on souhaite réaliser (filter les logs de prod, sur une machine donnée, pour un container donné, ...).

> **Remarque**
>
> Elan connaît le type et le nom de l'image grâce aux informations fournies dans le logback-spring.xml et plus particulièrement grâce aux lignes
>
> `xml <property scope="context" name="imageName" value="${APPLICATION_NAME}_back"/> <property scope="context" name="containerName" value="${APPLICATION_NAME}_back"/>`

Envoi des logs dans Marvel

> **Important**
>
> Cette partie n'a pas été faite sur Startix. Nous allons quand même voir que des solutions existent pour les projets qui souhaiteraient mettre en place la publication de leurs logs vers Marvel

Kubernetes

C'est la méthode recommandée à RTE, les projets étant encouragés à utiliser kubernetes plutôt qu'Ansible de façon générale.

Marvel n'est pas encore mis en place actuellement avec la version BareMetal de Kubernetes, mais il est prévu de le rajouter.

À terme, les projets sur iK8s n'auront qu'à sortir leurs logs dans la console pour pouvoir utiliser Marvel. Autrement dit, il suffira de faire en sorte que tous les logs de l'application soient en console et de faire les demandes sur Cosmos, kubernetes prendra en charge tout le reste.

Ansible

La documentation pour se connecter à Marvel pour les applications Ansible se trouve dans marvel_ansible.md.

Mise en place de MARVEL pour les projets utilisant Ansible

Introduction

MARVEL est une offre de service permettant de centraliser les logs d'une application. Plus d'informations sur MARVEL sont disponibles dans la documentation Marvel.

C'est le début de cette solution qui a été mis en place sur Startix. Seule la partie de récupération des logs via docker-compose a été faite, car il a été décidé que la mise en place de VMs pour illustrer la centralisation des logs n'était pas nécessaire et trop coûteux en temps.

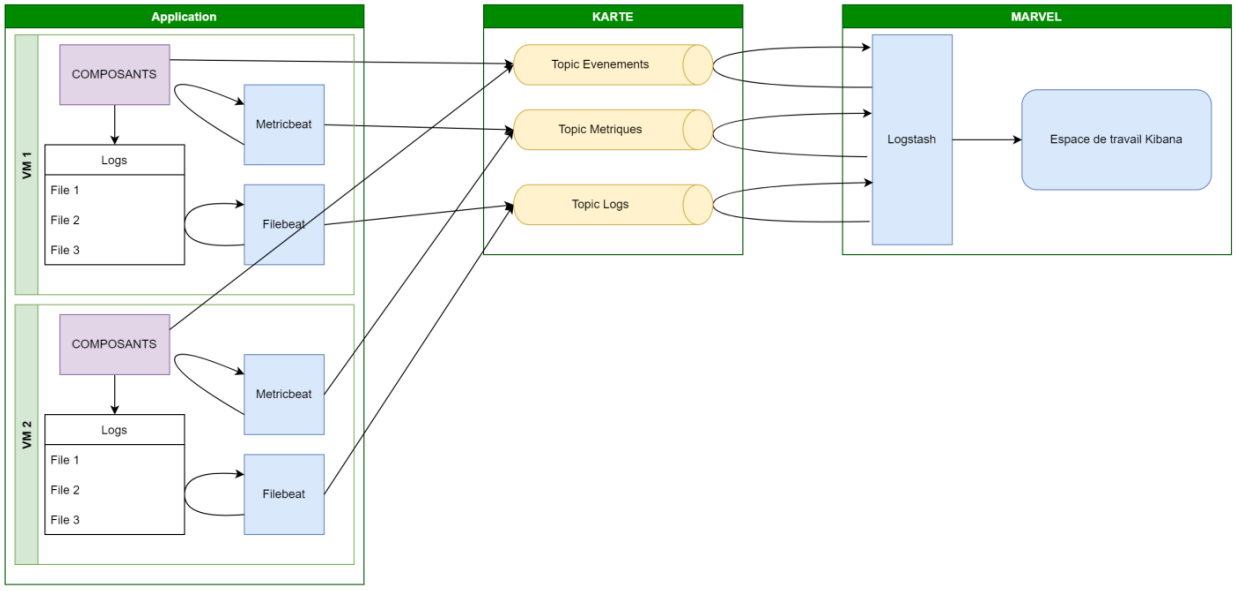
De plus, Kubernetes, qui est la méthode à privilégier, va prochainement mettre en place l'interfaçage avec Marvel pour les applications dont les logs sortent directement dans la console. Pour plus de détails, consulter la section dédiée.

> **Startix n'utilise donc pas filebeat ni K@RTE pour l'instant.**

Le projet smash-deploiement a mis en place la configuration pour se connecter à Marvel. Voici la documentation fournie par le projet sur laquelle Startix s'est appuyé.

Architecture cible

Voici l'architecture cible du projet :



La partie Application désigne le périmère de l'application. Pour SMASH: * L'application est constituée de plusieurs instances microservices déployés sur des container docker (ici: brique COMPOSANTS) répartis dans x VMs. * Chaque microservice produit des logs sur le FS du container, monté sur le FS de la VM hôte, dans des répertoires partagés entre tous les microservices: * sur Console * dans un fichier .log * dans un fichier .json * Un container Filebeat est déployé sur chaque VM, il lit les fichiers de logs .json de tous les microservices et les envoie sur un unique Topic K@RTE (par environnement) dédié à SMASH

La partie K@RTE désigne l'offre de service d'échange inter-applicatif (basé sur la technologie KAFKA): Offre de Services autour de KaRTE: * Elle héberge le Topic dédié aux logs de SMASH pour chaque environnement * Une une instance de l'offre K@RTE par environnement: OPF, PREPROD, PROD

La partie MARVEL désigne l'offre de service pour la centralisation et la corrélation des logs applicatifs Offre de service MARVEL: * Elle s'occupe de lire le Topic K@RTE recevant les logs de SMASH (toutes les instances de tous les microservices) avec le composant Logstash * Elle stocke ces logs dans un index. Pour SMASH, un index unique par environnement pour permettre une corrélation des logs de toutes les instances de tous les microservices, pour chaque environnement * Elle propose un espace de travail KIBANA pour requêter les logs (analyse) et construire des dashboards et indicateurs pour le suivi de l'application en RUN

Configuration de chaque microservice

Pom.xml

Dans le `pom.xml` du projet, rajouter

```
    <dependency>
        <groupId>com.rte_france.elan</groupId>
        <artifactId>elan-logger</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </dependency>
```

Remarque : En local, ne pas oublier de mettre ce bloc dans le `~/.m2/settings.xml`:

```
    <profile>
        <id>devin-nexus-snapshots</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <repositories>
          <!-- Repository permettant de telecharger des SNAPSHOTS -->
          <repository>
              <id>nexus</id>
              <name>nexus</name>
              <url>https://devin-depot.rte-france.com/repository/maven-snapshots/</url>
          </repository>
    </repositories>
</profile>
```

Si vous n'avez pas ce profile, vous n'arriverez pas à télécharger le snapshot de elan-logger. Cette dépendance fournit un provider au sens logback pour construire chaque évènement de log et l'enrichir avec les informations utiles à inclure dans les logs résultants.

Fichier logback-spring.xml

Ce fichier sert, pour chaque microservice, à paramétrer le fonctionnement des logs (format, variables globales ...):

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/defaults.xml"/>

    <!-- Spring Log properties -->
    <springProperty scope="context" name="springApplicationName" source="spring.application.name"/>
    <springProperty scope="context" name="springActiveProfile" source="spring.profiles.active"/>

    <!-- Log properties -->
    <property scope="context" name="imageName" value="SMASH_BACK"/>
    <property scope="context" name="containerName" value="SMASH_BACK"/>
    <property scope="context" name="vmIP" value="${HOST_IP}"/>
    <property scope="context" name="vmName" value="${HOST_NAME}"/>

    <!--Classic appender to log console-->
    <appender name="Console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX"} %-5level %class{36} %M %L - %msg%n</pattern>
        </encoder>
    </appender>

    <!-- Classic appender to log file-->
    <appender name="RollingFile" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>/logs/application.log</file>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <charset>UTF-8</charset>
            <pattern>%d{"yyyy-MM-dd'T'HH:mm:ss.SSSXXX", UTC} %-5level %M %class{0} %L - %msg%n</pattern>
        </encoder>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

            <!-- Rollover daily and when the file reaches 10 MegaBytes -->
            <fileNamePattern>/logs/application-%d{yyyy-MM-dd}.log.gz</fileNamePattern>
            <maxHistory>150</maxHistory>
        </rollingPolicy>
    </appender>

    <!-- Appender to log to file in a JSON format, one JSON object per line -->
    <appender name="JSON" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_PATH_LOGGER}/smash_back_application.json</file>
```

```
        <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
            <fileNamePattern>${LOG_PATH_LOGGER}/smash_back_application.%d{yyyy-MM-dd}.%i.json</fileNamePattern
            <maxFileSize>10MB</maxFileSize>
            <maxHistory>3</maxHistory>
            <totalSizeCap>20MB</totalSizeCap>
        </rollingPolicy>
        <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <provider class="com.rte_france.elan.logger.provider.LogDataJsonProvider">
                </provider>
            </providers>
        </encoder>
    </appender>

    <!-- LOG everything at INFO level -->
    <root level="info">
        <appender-ref ref="RollingFile" />
          <!-- Comment this appender to disable console logging in the Prod environment -->
          <appender-ref ref="Console" />
          <appender-ref ref="JSON"/>
    </root>
</configuration>
```

Ce fichier comporte: * Les variables globales utilisées par Spring pour les logs (IP, HOST,...) * Les appenders pour les différents modes de logs. Ici: * Console: pour les logs sur Console * RollingFile: pour les logs sauvegardés sur le FS dans les fichiers "application.log" * JSON: pour les logs sauvegardés sur le FS dans les fichiers "[NOM_MICROSERVICE]_application.json": * ce sont ceux-là qui seront récupérés par la suite par le composant Filebeat pour les envoyer à MARVEL en passant par le bus K@RTE * Cet appender se base sur le provider d'informations de log "com.rte_france.elan.logger.provider.LogDataJsonProvider" contenu dans la librairie elan-logger, ajoutée au pom.xml

Ce fichier est à mettre dans le projet de déploiement GIT sous le chemin : `smash-deploiement/livraison/config/logs/[PROJET]`, avec [PROJET] le nom du projet GIT de chaque microservice (par exemple "smash_back").

> **Variables d'environnement**
>
> D'après le fichier, on a besoin que les variables d'environnement suivantes soient définies: * HOST_IP: IP de la machine * HOST_NAME: Nom de la machine * LOG_PATH_LOGGER: chemin sur la VM, vu de l'intérieur de l'image Docker, du fichier de log JSON * LOGGING_CONFIG: chemin sur la VM, vu de l'intérieur de l'image Docker, du fichier de > configuration `logback-spring.xml`
>
> Ces variables sont définies dans le fichier env.conf du projet de déploiement devops

Configuration des variables d'environnement

Ajouter les définitions suivantes dans les fichiers env.conf du projet de déploiement, pour tous les environnements :

```
# environnement, utile pour séparer les logs par environnement
spring.profiles.active=demo

# Logging configuration path for each microservice
LOGGING_CONFIG=file:/config/logs/logback-spring.xml

# json log file path for log centralization
LOG_PATH_LOGGER=/logs/json

# Useful for the container, see docker-compose file, also for logging purposes
HOST_NAME = $(cat /etc/hostname)
HOST_IP=$(hostname -i)
```

Fichier docker-compose.yml

Dans le docker-compose.yml de chaque microservice ajouter les éléments suivants: les variables d'environnements HOST_NAME et HOST_IP

```
environment :
    - HOST_NAME=${HOST_NAME}
    - HOST_IP=${HOST_IP}
```

les mappings vers les différents fichiers de configuration et de logs

```
volumes: # /chemin/sur/la/vm:/chemin/dans/le/conteneur
    # ne pas mettre ici de chemin absolu impliquant /applis/{{projet}}/config car ce
    # lien peut ne pas avoir été créé au moment de l'utilisation du docker compose
    # (création du point de rollback bdd par exemple)
    - /applis/smash/logs/smash_back/:/logs/
```

```
    - /applis/smash/logs/all-json/:/logs/json/
    - /applis/smash/config/logs/smash_back/:/config/logs/
```

> **Attention**
>
> - Les différents mappings et variables d'environnement sont à ajouter, il ne faut pas > remplacer l'existant
> - Pour chaque mapping, la partie gauche décrit le chemin dédié sur la VM, la partie droite > le chemin vu de l'intérieur de l'image docker: > * /applis/smash/logs/smash_back/:/logs/ => où trouver les fichiers de logs application.log
> - /applis/smash/logs/all-json/:/logs/json/ => où trouver les fichiers de logs JSON
> - /applis/smash/config/logs/smash_back/:/config/logs/ => où trouver la configuration > `logback-spring.xml`
> - ATTENTION: remplacer smash_back par le nom du projet GIT de chaque microservice (par exemple "smash_back")

Vérification d'étape

On peut d'ors et déjà vérifier le fonctionnement de la première partie, à savoir que : * le microservice produit bien des logs au format JSON, dans le bon répertoire, avec le bon nom * les logs json sont bien renseignées par les informations de logs nécessaires

Configuration du filebeat

Configuration du rôle filebeat

Créer le rôle "filebeat" dans smash-deploiement/livraison/installation/roles: * Créer le fichier smash-deploiement/livraison/installation/roles/filebeat/tasks/main.yml, avec le contenu:

```
# Exemple de taches à exécuter sur le serveur filebeat

# Création du réportoire de logs pour le service filebeat - l'image filebeat est modifiée pour
# utiliser le compte du projet.
- name: "Création dossier : /applis/{{projet}}/logs/filebeat"
    file: path=/applis/{{projet}}/logs/filebeat state=directory recurse=yes mode='0775' owner={{user_projet}}

# Création du réportoire data pour le service filebeat - l'image filebeat est modifiée pour
# utiliser le compte du projet.
- name: "Création dossier : /applis/{{projet}}/data/filebeat"
    file: path=/applis/{{projet}}/data/filebeat state=directory recurse=yes mode='0775' owner={{user_projet}}

# Création du réportoire data pour le service filebeat - l'image filebeat est modifiée pour
# utiliser le compte du projet.
- name: "Création dossier : /applis/{{projet}}/logs/all-json"
    file: path=/applis/{{projet}}/logs/all-json state=directory recurse=yes mode='0775' owner={{user_projet}}
```

- Créer le fichier smash-deploiement/livraison/installation/roles/filebeat/is-active/tasks/main.yml, avec le contenu:

```
- name: "Controle que le {{services_def[\"filebeat\"].nom}} est actif"
    shell: "systemctl is-active {{services_def[\"filebeat\"].nom}}"
    register: command_result
    args:
      executable: /bin/bash
    # timeout: 3min
    retries: 36
    delay: 5
    until: command_result.rc == 0
    changed_when: false
```

Utilisation du rôle filebeat dans site.xml

- Insérer le code suivant dans le fichier smash-deploiement/livraison/installation/site.yml:

```
- hosts: filebeat-Group
  roles:
    - filebeat
```

- Insérer le code suivant dans le fichier smash-deploiement/livraison/installation/site.yml:

```
- hosts: filebeat-Group
  roles:
    - { role: common/service/create, services: ["filebeat"]}
    - { role: common/service/manage, service: "filebeat", command: "start" }
    - filebeat/is-active
```

> **La position dans le fichier est importante**
>
> - Les bouts de code précédents sont à insérer au bon endroit dans le fichier en question. Pour ce faire, se référer à la configuration des autres microservices
> - La position des - dans le yaml est très importante, il faut aussi bien respecter le nombre d'espaces ' ' avant les -

Modification de l'inventory

- Ajouter le code suivant dans le fichier smash-deploiement/livraison/installation/inventory/[environnement] de tous les environnements:

```
# Machines du groupe filebeat
[filebeat-Group]
vm1
vm2

# Enfants du groupe smash
[smash:children]
...
filebeat-Group
```

`vm1` et `vm2` sont données ici à titre indicatif, utiliser les bons noms de vm définis au début du même fichier

Modification du fichier group_vars/all

- Modifier le fichier smash-deploiement/livraison/installation/inventory/group_vars/all en ajoutant le code suivant:

```
"filebeat":
  nom: "smash-filebeat.service"
  composeFile: "docker-compose-filebeat.yml"
  rollbackContainer: "smash-filebeat"
```

Attention à respecter les espaces ' ' avant chaque ligne et l'indentation

Création du fichier de service smash-filebeat.service

Créer, dans le répertoire smash-deploiement/livraison/config/service/ le fichier "smash-filebeat.service" avec le contenu suivant:

```
[Unit]
Description=Service smash-filebeat
After=docker.service
Requires=docker.service
Wants=systemd-hostnamed.service loadenv.service

[Service]
User=docker
Restart=always
# Chemin du fichier données sensibles (ex : password) de l'environnement.
EnvironmentFile=/applis/livraisons/private.conf

# En savoir plus : voir le role 'roles/common/activation' pour la création du lien symbolique 'config'
Environment="COMPOSE_FILE=/applis/smash/config/docker-compose-filebeat.yml"

=-/usr/bin/docker compose -f ${COMPOSE_FILE} stop
ExecStartPre=-/usr/bin/docker compose -f ${COMPOSE_FILE} rm -fv

# On controle que l'image est bien disponible et que le container peut être créé
# create --no-recreate : Permet de creer le container sans le démarrer, afin de vérifier que l'image est
disponible (en local ou sur inca)
# A remplacer dans les prochaines versions de docker par : up --no-start
# on n'utilise pas pull car ca ne fonctionne pas en dev si l'image n'est pas sur inca
# On créer aussi le network, car il n'est pas créé automatiquement
ExecStartPre=-/bin/docker network create config_default
```

```
ExecStartPre=/usr/bin/docker compose -f ${COMPOSE_FILE} create --no-recreate

ExecStart=/usr/bin/docker compose -f ${COMPOSE_FILE} up
ExecStop=/usr/bin/docker compose -f ${COMPOSE_FILE} stop
ExecReload=/usr/bin/docker compose -f ${COMPOSE_FILE} restart

[Install]
WantedBy=multi-user.target
```

> **Oracle linux 8**
> Attention, le fichier est adapté à une VM Oracle Linux 8 qui est la cible de SMASH

Construction d'une image filebeat dans INCA du projet

Sur une VM sur laquelle un client docker est installé, créer un fichier Dockerfile contenant:

```
FROM inca.rte-france.com/rtsi/filebeat:8.2.2
ENV TZ=Europe/Paris LANG=C.UTF-8
LABEL maintainer="Equipe SMASH"

USER root
RUN set -x \
  # create smash user/group
  && groupadd -g 2000 smash \
  && useradd -g smash -s /sbin/nologin -u 2000 smash \
  && chown -R smash:smash /usr/share/filebeat

USER smash
CMD /usr/share/filebeat/filebeat
```

> **Attention**
>
> - inca.rte-france.com/rtsi/filebeat:8.2.2 est une image disponible sur INCA RTSI
> - le user et group smash (2000) doit être créé de manière uniforme avec les autres microservices

Dans le même répertoire contenant le fichier Dockerfile qui vient d'être créé, lancer la commande:

```
docker build -t filebeat:8.2.2 .
```

Tagger l'image:

```
docker tag filebeat:8.2.2 inca.rte-france.com/smash/smash-filebeat:8.2.2
```

Push l'image sur INCA projet:

```
docker login inca.rte-france.com

docker push inca.rte-france.com/smash/smash-filebeat:8.2.2
docker logout inca.rte-france.com
```

> **login docker**
> saisir le username (nom + 3 premières lettres du prénom) mot de passe: mot de passe windows

TODO: Chercher comment bloquer et débloquer la suppression de l'image dans INCA. En effet, la politique de rétention peut avoir comme effet de supprimer l'image, ce qui empêchera la récupération d'une image filebeat et le lancement de celui-ci.

Création du fichier de configuration de filebeat

Créer dans le répertoire "smash-deploiement/livraison/config/environnement/[environnement]/filebeat" pour chacun des environnements (demo, pfri, prod...), le fichier filebeat.yml, avec le contenu:

```
########################## Filebeat Configuration #############################
# This file is a full configuration example documenting all non-deprecated
# options in comments. For a shorter configuration example, that contains only
# the most common options, please see filebeat.yml in the same directory.
#
# You can find the full configuration reference here:
# https://www.elastic.co/guide/en/beats/filebeat/index.html


#========================== Modules configuration =============================
```

```
#========================== Filebeat inputs =============================


# List of inputs to fetch data.
filebeat.inputs:
# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

# Type of the files. Based on this the way the file is read is decided.
# The different types cannot be mixed in one input
#
# Possible options are:
# * log: Reads every line of the log file (default)
# * stdin: Reads the standard in

#------------------------------ Log input --------------------------------
- type: log

# Change to true to enable this input configuration.
enabled: true

# Paths that should be crawled and fetched. Glob based paths.
# To fetch all ".log" files from a specific level of subdirectories
# /var/log/*/*.log can be used.
# For each file found under this path, a harvester is started.
# Make sure not file is defined twice as this can lead to unexpected behaviour.
paths:
  - /var/log/all-json/*.json
  #- c:\programdata\elasticsearch\logs\*

# Configure the file encoding for reading files with international characters
# following the W3C recommendation for HTML5 (http://www.w3.org/TR/encoding).
# Some sample encodings:
#   plain, utf-8, utf-16be-bom, utf-16be, utf-16le, big5, gb18030, gbk,
#   hz-gb-2312, euc-kr, euc-jp, iso-2022-jp, shift-jis, ...
#encoding: plain

# Exclude lines. A list of regular expressions to match. It drops the lines that are
# matching any regular expression from the list. The include_lines is called before
# exclude_lines. By default, no lines are dropped.
#exclude_lines: ['^DBG']

# Include lines. A list of regular expressions to match. It exports the lines that are
# matching any regular expression from the list. The include_lines is called before
# exclude_lines. By default, all the lines are exported.
#include_lines: ['^ERR', '^WARN']

# Exclude files. A list of regular expressions to match. Filebeat drops the files that
# are matching any regular expression from the list. By default, no files are dropped.
#exclude_files: ['.gz$']

# Optional additional fields. These fields can be freely picked
# to add additional information to the crawled log files for filtering
# fields:
#   level: debug
#   review: 1

# Set to true to store the additional fields as top level fields instead
# of under the "fields" sub-dictionary. In case of name conflicts with the
# fields added by Filebeat itself, the custom fields overwrite the default
# fields.
#fields_under_root: false

# Set to true to publish fields with null values in events.
#keep_null: false
# By default, all events contain `host.name`. This option can be set to true
# to disable the addition of this field to all events. The default value is
# false.
#publisher_pipeline.disable_host: false

# Ignore files which were modified more then the defined timespan in the past.
# ignore_older is disabled by default, so no files are ignored by setting it to 0.
# Time strings like 2h (2 hours), 5m (5 minutes) can be used.
#ignore_older: 0
```

```
# How often the input checks for new files in the paths that are specified
# for harvesting. Specify 1s to scan the directory as frequently as possible
# without causing Filebeat to scan too frequently. Default: 10s.
#scan_frequency: 10s

# Defines the buffer size every harvester uses when fetching the file
#harvester_buffer_size: 16384

# Maximum number of bytes a single log event can have
# All bytes after max_bytes are discarded and not sent. The default is 10MB.
# This is especially useful for multiline log messages which can get large.
#max_bytes: 10485760

# Characters which separate the lines. Valid values: auto, line_feed, vertical_tab, form_feed,
# carriage_return, carriage_return_line_feed, next_line, line_separator, paragraph_separator.
#line_terminator: auto

### Recursive glob configuration

# Expand "**" patterns into regular glob patterns.
#recursive_glob.enabled: true

### JSON configuration

# Decode JSON options. Enable this if your logs are structured in JSON.
# JSON key on which to apply the line filtering and multiline settings. This key
# must be top level and its value must be string, otherwise it is ignored. If
# no text key is defined, the line filtering and multiline features cannot be used.
#json.message_key:

# By default, the decoded JSON is placed under a "json" key in the output document.
# If you enable this setting, the keys are copied top level in the output document.
json.keys_under_root: true

# If keys_under_root and this setting are enabled, then the values from the decoded
# JSON object overwrite the fields that Filebeat normally adds (type, source, offset, etc.)
# in case of conflicts.
json.overwrite_keys: true

# If this setting is enabled, Filebeat adds a "error.message" and "error.key: json" key in case of JSON
# unmarshaling errors or when a text key is defined in the configuration but cannot
# be used.
json.add_error_key: true

### Multiline options

# Multiline can be used for log messages spanning multiple lines. This is common
# for Java Stack Traces or C-Line Continuation
# The regexp Pattern that has to be matched. The example pattern matches all lines starting with [
#multiline.pattern: ^\[

# Defines if the pattern set under pattern should be negated or not. Default is false.
#multiline.negate: false

# Match can be set to "after" or "before". It is used to define if lines should be append to
# a pattern# that was (not) matched before or after or as long as a pattern is not matched based on negate.
# Note: After is the equivalent to previous and before is the equivalent to to next in Logstash
#multiline.match: after

# The maximum number of lines that are combined to one event.
# In case there are more the max_lines the additional lines are discarded.
# Default is 500
#multiline.max_lines: 500

# After the defined timeout, an multiline event is sent even if no new pattern was found to start a new event
# Default is 5s.
#multiline.timeout: 5s

# Setting tail_files to true means filebeat starts reading new files at the end
# instead of the beginning. If this is used in combination with log rotation
# this can mean that the first entries of a new file are skipped.
#tail_files: false

# The Ingest Node pipeline ID associated with this input. If this is set, it
```

```
# overwrites the pipeline option from the Elasticsearch output.
#pipeline:

# If symlinks is enabled, symlinks are opened and harvested. The harvester is opening the
# original for harvesting but will report the symlink name as source.
#symlinks: false

# Backoff values define how aggressively filebeat crawls new files for updates
# The default values can be used in most cases. Backoff defines how long it is waited
# to check a file again after EOF is reached. Default is 1s which means the file
# is checked every second if new lines were added. This leads to a near real time crawling.
# Every time a new line appears, backoff is reset to the initial value.
#backoff: 1s

# Max backoff defines what the maximum backoff time is. After having backed off multiple times
# from checking the files, the waiting time will never exceed max_backoff independent of the
# backoff factor. Having it set to 10s means in the worst case a new line can be added to a log
# file after having backed off multiple times, it takes a maximum of 10s to read the new line
#max_backoff: 10s

# The backoff factor defines how fast the algorithm backs off. The bigger the backoff factor,
# the faster the max_backoff value is reached. If this value is set to 1, no backoff will happen.
# The backoff value will be multiplied each time with the backoff_factor until max_backoff is reached
#backoff_factor: 2

# Max number of harvesters that are started in parallel.
# Default is 0 which means unlimited
#harvester_limit: 0

### Harvester closing options
# Close inactive closes the file handler after the predefined period.
# The period starts when the last line of the file was, not the file ModTime.
# Time strings like 2h (2 hours), 5m (5 minutes) can be used.
#close_inactive: 5m

# Close renamed closes a file handler when the file is renamed or rotated.
# Note: Potential data loss. Make sure to read and understand the docs for this option.
#close_renamed: false

# When enabling this option, a file handler is closed immediately in case a file can't be found
# any more. In case the file shows up again later, harvesting will continue at the last known position
# after scan_frequency.
#close_removed: true

# Closes the file handler as soon as the harvesters reaches the end of the file.
# By default this option is disabled.
# Note: Potential data loss. Make sure to read and understand the docs for this option.
#close_eof: false

### State options
# Files for the modification data is older then clean_inactive the state from the registry is removed

# By default this is disabled.
#clean_inactive: 0

# Removes the state for file which cannot be found on disk anymore immediately
#clean_removed: true

# Close timeout closes the harvester after the predefined time.
# This is independent if the harvester did finish reading the file or not.
# By default this option is disabled.
# Note: Potential data loss. Make sure to read and understand the docs for this option.
#close_timeout: 0

# Defines if inputs is enabled
#enabled: true

# ========================== Filebeat global options ============================

# Registry data path. If a relative path is used, it is considered relative to the
# data path.
#filebeat.registry.path: ${path.data}/registry

# The permissions mask to apply on registry data, and meta files. The default
# value is 0600. Must be a valid Unix-style file permissions mask expressed in
```

```
# octal notation. This option is not supported on Windows.
#filebeat.registry.file_permissions: 0600

# The timeout value that controls when registry entries are written to disk
# (flushed). When an unwritten update exceeds this value, it triggers a write
# to disk. When flush is set to 0s, the registry is written to disk after each
# batch of events has been published successfully. The default value is 0s.
#filebeat.registry.flush: 0s

# Starting with Filebeat 7.0, the registry uses a new directory format to store
# Filebeat state. After you upgrade, Filebeat will automatically migrate a 6.x
# registry file to use the new directory format. If you changed
# filebeat.registry.path while upgrading, set filebeat.registry.migrate_file to
# point to the old registry file.
#filebeat.registry.migrate_file: ${path.data}/registry

# By default Ingest pipelines are not updated if a pipeline with the same ID
# already exists. If this option is enabled Filebeat overwrites pipelines
# everytime a new Elasticsearch connection is established.
#filebeat.overwrite_pipelines: false

# How long filebeat waits on shutdown for the publisher to finish.
# Default is 0, not waiting.
#filebeat.shutdown_timeout: 0

# Enable filebeat config reloading
#filebeat.config:
#  inputs:
#    enabled: false
#    path: inputs.d/*.yml
#    reload.enabled: true
#    reload.period: 10s
#  modules:
#    enabled: false
#    path: modules.d/*.yml
#    reload.enabled: true
#    reload.period: 10s

# ================================== General ===================================

# The name of the shipper that publishes the network data. It can be used to group
# all the transactions sent by a single shipper in the web interface.
# If this options is not defined, the hostname is used.
#name:

# The tags of the shipper are included in their own field with each
# transaction published. Tags make it easy to group servers by different
# logical properties.
#tags: ["service-X", "web-tier"]

# Optional fields that you can specify to add additional information to the
# output. Fields can be scalar values, arrays, dictionaries, or any nested
# combination of these.
#fields:
#  env: staging

# If this option is set to true, the custom fields are stored as top-level
# fields in the output document instead of being grouped under a fields
# sub-dictionary. Default is false.
#fields_under_root: false

# Internal queue configuration for buffering events to be published.
#queue:
#  Queue type by name (default 'mem')
#  The memory queue will present all available events (up to the outputs
#  bulk_max_size) to the output, the moment the output is ready to server
#  another batch of events.
#  mem:
     #Max number of events the queue can buffer.
#    events: 4096

    #Hints the minimum number of events stored in the queue,
    #before providing a batch of events to the outputs.
    #The default value is set to 2048.
    #A value of 0 ensures events are immediately available
```

```
    #to be sent to the outputs.
#   flush.min_events: 2048

    #Maximum duration after which events are available to the outputs,
    #if the number of events stored in the queue is < `flush.min_events`.
#   flush.timeout: 1s

    # The spool queue will store events in a local spool file, before
    # forwarding the events to the outputs.
    #
    # Beta: spooling to disk is currently a beta feature. Use with care.
    #
    # The spool file is a circular buffer, which blocks once the file/buffer is full.
    # Events are put into a write buffer and flushed once the write buffer
    # is full or the flush_timeout is triggered.
    # Once ACKed by the output, events are removed immediately from the queue,
    # making space for new events to be persisted.
    #spool:
      # The file namespace configures the file path and the file creation settings.
      # Once the file exists, the `size`, `page_size` and `prealloc` settings
      # will have no more effect.
      #file:
        # Location of spool file. The default value is ${path.data}/spool.dat.
        #path: "${path.data}/spool.dat"

        # Configure file permissions if file is created. The default value is 0600.
        #permissions: 0600

        # File size hint. The spool blocks, once this limit is reached. The default value is 100 MiB.
        #size: 100MiB

        # The files page size. A file is split into multiple pages of the same size. The default value is 4KiB
        #page_size: 4KiB

        # If prealloc is set, the required space for the file is reserved using
        # truncate. The default value is true.
        #prealloc: true

      # Spool writer settings
      # Events are serialized into a write buffer. The write buffer is flushed if:
      # - The buffer limit has been reached.# - The configured limit of buffered events is reached.
      # - The flush timeout is triggered.
      #write:
        # Sets the write buffer size.
        #buffer_size: 1MiB

        # Maximum duration after which events are flushed if the write buffer
        # is not full yet. The default value is 1s.
        #flush.timeout: 1s

        # Number of maximum buffered events. The write buffer is flushed once the
        # limit is reached.
        #flush.events: 16384

        # Configure the on-disk event encoding. The encoding can be changed
        # between restarts.
        # Valid encodings are: json, ubjson, and cbor.
        #codec: cbor

      #read:
        # Reader flush timeout, waiting for more events to become available, so
        # to fill a complete batch as required by the outputs.
        # If flush_timeout is 0, all available events are forwarded to the
        # outputs immediately.
        # The default value is 0s.
        #flush.timeout: 0s

# Sets the maximum number of CPUs that can be executing simultaneously. The
# default is the number of logical CPUs available in the system.
#max_procs:


# ================================== Outputs ===================================

# Configure what output to use when sending the data collected by the beat.
```

```
# ------------------------------ Logstash Output -------------------------------
output.logstash:
  # Boolean flag to enable or disable the output module.
  enabled: false

  # The Logstash hosts
  hosts: ["PF9SOREEMMA011.applispfref.sipfref.local:5044"]

  # https://mostafa-asg.github.io/posts/publishing-text-messages-to-the-kafka-without-writing-any-code/
  #codec.format:
    # string: '%{[message]}'

  # Number of workers per Logstash host.
  #worker: 1

  # Set gzip compression level.
  #compression_level: 3

  # Configure escaping HTML symbols in strings.
  #escape_html: false

  # Optional maximum time to live for a connection to Logstash, after which the
  # connection will be re-established. A value of `0s` (the default) will
  # disable this feature.
  #
  # Not yet supported for async connections (i.e. with the "pipelining" option set)
  #ttl: 30s

  # Optionally load-balance events between Logstash hosts. Default is false.
  #loadbalance: false

  # Number of batches to be sent asynchronously to Logstash while processing
  # new batches.
  #pipelining: 2

  # If enabled only a subset of events in a batch of events is transferred per
  # transaction. The number of events to be sent increases up to `bulk_max_size`
  # if no error is encountered.
  #slow_start: false


  # The number of seconds to wait before trying to reconnect to Logstash
  # after a network error. After waiting backoff.init seconds, the Beat
  # tries to reconnect. If the attempt fails, the backoff timer is increased
  # exponentially up to backoff.max. After a successful connection, the backoff
  # timer is reset. The default is 1s.
  #backoff.init: 1s


  # The maximum number of seconds to wait before attempting to connect to
  # Logstash after a network error. The default is 60s.
  #backoff.max: 60s

  # Optional index name. The default index name is set to filebeat
  # in all lowercase.
  #index: 'filebeat'

  # SOCKS5 proxy server URL
  #proxy_url: socks5://user:password@socks5-server:2233

  # Resolve names locally when using a proxy server. Defaults to false.
  #proxy_use_local_resolver: false

  # Enable SSL support. SSL is automatically enabled if any SSL setting is set.
  #ssl.enabled: true

  # Configure SSL verification mode. If `none` is configured, all server hosts
  # and certificates will be accepted. In this mode, SSL based connections are
  # susceptible to man-in-the-middle attacks. Use only for testing. Default is
  # `full`.
  #ssl.verification_mode: full

  # List of supported/valid TLS versions. By default all TLS versions from 1.1
  # up to 1.3 are enabled.
```

```
  #ssl.supported_protocols: [TLSv1.1, TLSv1.2, TLSv1.3]

  # Optional SSL configuration options. SSL is off by default.
  # List of root certificates for HTTPS server verifications
  #ssl.certificate_authorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client certificate key
  #ssl.key: "/etc/pki/client/cert.key"

  # Optional passphrase for decrypting the Certificate Key.
  #ssl.key_passphrase: ''

  # Configure cipher suites to be used for SSL connections
  #ssl.cipher_suites: []

  # Configure curve types for ECDHE-based cipher suites
  #ssl.curve_types: []

  # Configure what types of renegotiation are supported. Valid options are
  # never, once, and freely. Default is never.
  #ssl.renegotiation: never

  # Configure a pin that can be used to do extra validation of the verified certificate chain,
  # this allow you to ensure that a specific certificate is used to validate the chain of trust.
  #
  # The pin is a base64 encoded string of the SHA-256 fingerprint.
  #ssl.ca_sha256: ""

  # The number of times to retry publishing an event after a publishing failure.
  # After the specified number of retries, the events are typically dropped.
  # Some Beats, such as Filebeat and Winlogbeat, ignore the max_retries setting
  # and retry until all events are published. Set max_retries to a value less
  # than 0 to retry until all events are published. The default is 3.
  #max_retries: 3

  # The maximum number of events to bulk in a single Logstash request. The
  # default is 2048.
  #bulk_max_size: 2048

  # The number of seconds to wait for responses from the Logstash server before
  # timing out. The default is 30s.
  #timeout: 30s

# ------------------------------ Kafka Output -------------------------------

output.kafka:
  # Boolean flag to enable or disable the output module.
  enabled: true

  # https://mostafa-asg.github.io/posts/publishing-text-messages-to-the-kafka-without-writing-any-code/
  #codec.format:
  # string: '%{[message]}'

  # The list of Kafka broker addresses from which to fetch the cluster metadata.
  # The cluster metadata contain the actual Kafka brokers events are published
  # to.
  hosts: ["<FQDN_KARTE>:<PORT_KARTE>"]

  # The Kafka topic used for produced events. The setting can be a format string
  # using any event field. To set the topic from document type use `%{[type]}`.
  topic: '<TOPIC_KARTE>'

  # The Kafka event key setting. Use format string to create a unique event key.
  # By default no event key will be generated.
  #key: ''

  # The Kafka event partitioning strategy. Default hashing strategy is `hash`
  # using the `output.kafka.key` setting or randomly distributes events if
  # `output.kafka.key` is not configured.
  #partition.hash:
    # If enabled, events will only be published to partitions with reachable
    # leaders. Default is false.
```

```
    #reachable_only: false

    # Configure alternative event field names used to compute the hash value.
    # If empty `output.kafka.key` setting will be used.
    # Default value is empty list.
    #hash: []

  # Authentication details. Password is required if username is set. PLAIN Auth'
  username: '<USERNAME_KARTE>'
  password: '<PASSWORD_KARTE>'

  # SASL authentication mechanism used. Can be one of PLAIN, SCRAM-SHA-256 or SCRAM-SHA-512.
  # Defaults to PLAIN when `username` and `password` are configured.
  sasl.mechanism: 'SCRAM-SHA-256'

  # Kafka version Filebeat is assumed to run against. Defaults to the "1.0.0".
  #version: '1.0.0'

  # Configure JSON encoding
  #codec.json:
    # Pretty-print JSON event
    #pretty: false

    # Configure escaping HTML symbols in strings.
    #escape_html: false

  # Metadata update configuration. Metadata contains leader information
  # used to decide which broker to use when publishing.
  #metadata:
    # Max metadata request retry attempts when cluster is in middle of leader
    # election. Defaults to 3 retries.
    #retry.max: 3

    # Wait time between retries during leader elections. Default is 250ms.
    #retry.backoff: 250ms

    # Refresh metadata interval. Defaults to every 10 minutes.
    #refresh_frequency: 10m
    # Strategy for fetching the topics metadata from the broker. Default is false.
    #full: false

  # The number of concurrent load-balanced Kafka output workers.
  #worker: 1

  # The number of times to retry publishing an event after a publishing failure.
  # After the specified number of retries, events are typically dropped.
  # Some Beats, such as Filebeat, ignore the max_retries setting and retry until
  # all events are published. Set max_retries to a value less than 0 to retry
  # until all events are published. The default is 3.
  #max_retries: 3

  # The number of seconds to wait before trying to republish to Kafka
  # after a network error. After waiting backoff.init seconds, the Beat
  # tries to republish. If the attempt fails, the backoff timer is increased
  # exponentially up to backoff.max. After a successful publish, the backoff
  # timer is reset. The default is 1s.
  #backoff.init: 1s

  # The maximum number of seconds to wait before attempting to republish to
  # Kafka after a network error. The default is 60s.
  #backoff.max: 60s

  # The maximum number of events to bulk in a single Kafka request. The default
  # is 2048.
  #bulk_max_size: 2048

  # Duration to wait before sending bulk Kafka request. 0 is no delay. The default
  # is 0.
  #bulk_flush_frequency: 0s

  # The number of seconds to wait for responses from the Kafka brokers before
  # timing out. The default is 30s.
  #timeout: 30s

  # The maximum duration a broker will wait for number of required ACKs. The
```

```
# default is 10s.
#broker_timeout: 10s

# The number of messages buffered for each Kafka broker. The default is 256.
#channel_buffer_size: 256

# The keep-alive period for an active network connection. If 0s, keep-alives
# are disabled. The default is 0 seconds.
#keep_alive: 0

# Sets the output compression codec. Must be one of none, snappy and gzip. The
# default is gzip.
#compression: gzip

# Set the compression level. Currently only gzip provides a compression level
# between 0 and 9. The default value is chosen by the compression algorithm.
#compression_level: 4

# The maximum permitted size of JSON-encoded messages. Bigger messages will be
# dropped. The default value is 1000000 (bytes). This value should be equal to
# or less than the broker's message.max.bytes.
max_message_bytes: 1000000

# The ACK reliability level required from broker. 0=no response, 1=wait for
# local commit, -1=wait for all replicas to commit. The default is 1. Note:
# If set to 0, no ACKs are returned by Kafka. Messages might be lost silently
# on error.
required_acks: 1

# The configurable ClientID used for logging, debugging, and auditing
# purposes. The default is "beats".
client_id: "<CLIENT_ID>"

# Enable SSL support. SSL is automatically enabled if any SSL setting is set.
ssl.enabled: false

# Optional SSL configuration options. SSL is off by default.
# List of root certificates for HTTPS server verifications
# ssl.certificate_authorities: ["/saslssl/kafka.server.truststore.crt"]

# Configure SSL verification mode. If `none` is configured, all server hosts
# and certificates will be accepted. In this mode, SSL based connections are
# susceptible to man-in-the-middle attacks. Use only for testing. Default is
# `full`.
ssl.verification_mode: full

# List of supported/valid TLS versions. By default all TLS versions from 1.1
# up to 1.3 are enabled.
#ssl.supported_protocols: [TLSv1.1, TLSv1.2, TLSv1.3]

# Certificate for SSL client authentication
#ssl.certificate: "/etc/pki/client/cert.pem"

# Client Certificate Key
#ssl.key: "/etc/pki/client/cert.key"

# Optional passphrase for decrypting the Certificate Key.
#ssl.key_passphrase: ''

# Configure cipher suites to be used for SSL connections
#ssl.cipher_suites: []

# Configure curve types for ECDHE-based cipher suites
#ssl.curve_types: []

# Configure what types of renegotiation are supported. Valid options are
# never, once, and freely. Default is never.
#ssl.renegotiation: never

# Enable Kerberos support. Kerberos is automatically enabled if any Kerberos setting is set.
#kerberos.enabled: true

# Authentication type to use with Kerberos. Available options: keytab, password.
#kerberos.auth_type: password
```

```
  # Path to the keytab file. It is used when auth_type is set to keytab.
  #kerberos.keytab: /etc/security/keytabs/kafka.keytab

  # Path to the Kerberos configuration.
  #kerberos.config_path: /etc/krb5.conf

  # The service name. Service principal name is contructed from
  # service_name/hostname@realm.
  #kerberos.service_name: kafka

  # Name of the Kerberos user.
  #kerberos.username: elastic

  # Password of the Kerberos user. It is used when auth_type is set to password.
  #kerberos.password: changeme

  # Kerberos realm.
  #kerberos.realm: ELASTIC

# ================================== Paths =====================================

# The home path for the Filebeat installation. This is the default base path
# for all other path settings and for miscellaneous files that come with the
# distribution (for example, the sample dashboards).
# If not set by a CLI flag or in the configuration file, the default for the
# home path is the location of the binary.
#path.home:

# The configuration path for the Filebeat installation. This is the default
# base path for configuration files, including the main YAML configuration file
# and the Elasticsearch template file. If not set by a CLI flag or in the
# configuration file, the default for the configuration path is the home path.
#path.config: ${path.home}

# The data path for the Filebeat installation. This is the default base path
# for all the files in which Filebeat needs to store its data. If not set by a
# CLI flag or in the configuration file, the default for the data path is a data
# subdirectory inside the home path.
#path.data: ${path.home}/data

# The logs path for a Filebeat installation. This is the default location for
# the Beat's log files. If not set by a CLI flag or in the configuration file,
# the default for the logs path is a logs subdirectory inside the home path.
#path.logs: ${path.home}/logs

# ================================== Keystore ==================================

# Location of the Keystore containing the keys and their sensitive values.
#keystore.path: "${path.config}/beats.keystore"

# ================================== Template ==================================

# A template is used to set the mapping in Elasticsearch
# By default template loading is enabled and the template is loaded.
# These settings can be adjusted to load your own template or overwrite existing ones.

# Set to false to disable template loading.
#setup.template.enabled: true

# Template name. By default the template name is "filebeat-%{[agent.version]}"
# The template name and pattern has to be set in case the Elasticsearch index pattern is modified.
#setup.template.name: "filebeat-%{[agent.version]}"

# Template pattern. By default the template pattern is "-%{[agent.version]}-*" to apply to the default index
settings.
# The first part is the version of the beat and then -* is used to match all daily indices.
# The template name and pattern has to be set in case the Elasticsearch index pattern is modified.
#setup.template.pattern: "filebeat-%{[agent.version]}-*"

# Path to fields.yml file to generate the template
#setup.template.fields: "${path.config}/fields.yml"
# A list of fields to be added to the template and Kibana index pattern. Also
# specify setup.template.overwrite: true to overwrite the existing template.
#setup.template.append_fields:
#- name: field_name
```

```
#  type: field_type

# Enable JSON template loading. If this is enabled, the fields.yml is ignored.
#setup.template.json.enabled: false

# Path to the JSON template file
#setup.template.json.path: "${path.config}/template.json"

# Name under which the template is stored in Elasticsearch
#setup.template.json.name: ""

# Overwrite existing template
#setup.template.overwrite: false

# Elasticsearch template settings
setup.template.settings:

# A dictionary of settings to place into the settings.index dictionary
# of the Elasticsearch template. For more details, please check# https://www.elastic.co/guide/en/elasticsearch
#index:
  #number_of_shards: 1
  #codec: best_compression

# A dictionary of settings for the _source field. For more details, please check
# https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-source-field.html
#_source:
  #enabled: false

# ================================== Logging ===================================

# There are four options for the log output: file, stderr, syslog, eventlog
# The file output is the default.

# Sets log level. The default log level is info.
# Available log levels are: error, warning, info, debug
logging.level: info

# Enable debug output for selected components. To enable all selectors use ["*"]
# Other available selectors are "beat", "publish", "service"
# Multiple selectors can be chained.
#logging.selectors: [ ]

# Send all logging output to stderr. The default is false.
#logging.to_stderr: false

# Send all logging output to syslog. The default is false.
#logging.to_syslog: false

# Send all logging output to Windows Event Logs. The default is false.
#logging.to_eventlog: false

# If enabled, Filebeat periodically logs its internal metrics that have changed
# in the last period. For each metric that changed, the delta from the value at
# the beginning of the period is logged. Also, the total values for
# all non-zero internal metrics are logged on shutdown. The default is true.
logging.metrics.enabled: true

# The period after which to log the internal metrics. The default is 30s.
logging.metrics.period: 30s

# Logging to rotating files. Set logging.to_files to false to disable logging to
# files.
logging.to_files: true
logging.files:
  # Configure the path where the logs are written. The default is the logs directory
  # under the home path (the binary location).
  #path: /var/log/filebeat

  # The name of the files where the logs are written to.
  name: filebeat.log

  # Configure log file size limit. If limit is reached, log file will be
  # automatically rotated
  rotateeverybytes: 10485760 # = 10MB
```

```
  # Number of rotated log files to keep. Oldest files will be deleted first.
  keepfiles: 7

  # The permissions mask to apply when rotating log files. The default value is 0600.
  # Must be a valid Unix-style file permissions mask expressed in octal notation.
  #permissions: 0600

  # Enable log file rotation on time intervals in addition to size-based rotation.
  # Intervals must be at least 1s. Values of 1m, 1h, 24h, 7*24h, 30*24h, and 365*24h
  # are boundary-aligned with minutes, hours, days, weeks, months, and years as
  # reported by the local system clock. All other intervals are calculated from the
  # Unix epoch. Defaults to disabled.
  interval: 24h

  # Rotate existing logs on startup rather than appending to the existing
  # file. Defaults to true.
  # rotateonstartup: true

# Set to true to log messages in JSON format.
#logging.json: false

# Set to true, to log messages with minimal required Elastic Common Schema (ECS)
# information. Recommended to use in combination with `logging.json=true`
# Defaults to false.
#logging.ecs: false
```

- : adresse FQDN du serveur K@RTE. Ex: karte-int.rte-france.com pour l'OPF
- : port K@RTE. Ex: 9092 pour l'OPF
- : Nom du TOPIC K@RTE pour la centralisation des logs
- Dans la section 'output.kafka':
- username: ''
- password: ''
- et resp. les user et mdp pour l'accès à K@RTE
- : Client ID pour l'accès à K@RTE

> **Informations K@RTE**
> Ces informations sont fournies par K@RTE

Modification des droits pour permettre à Filebeat la lecture de son fichier de configuration

- Modifier le rôle common/installation/tasks/main.yml et modifier la task nommée "name: "Copy version to server "" pour ouvrir les droits en lecture sur le fichier de configuration filebeat.yml:

```
# Installation de la version
- name: "Copy version to server "
    copy: >
        src=../config
        dest=/applis/livraisons/{{version}}/
        owner={{user_admin}}
        group={{group_admin}}
        mode='0755'
        directory_mode='0755'
```

Mettre directory_mode='0755' comme ci-dessus

- Modifier, dans le même rôle la task nommée "name: "Change rights for {{user_projet}} "" comme suit:

```
  - name: "Change rights for {{user_projet}} "
    file: path=/applis/livraisons/{{version}}/config/environnement/{{env}}/ owner={{user_projet}} group={{grou
```

Mettre mode='0755' comme ci-dessus

Connexion à MARVEL

Prérequis K@RTE (TOPICS, infos...)

Afin d'anticiper, le temps de dérouler le mode opératoire en cours, demander à MARVEL de fournir pour chaque environnement (opf, pfri, prod...), les informations suivantes: * : adresse FQDN du serveur K@RTE. Ex: karte-int.rte-france.com pour l'OPF * : port K@RTE. Ex: 9092 pour l'OPF * : Nom du TOPIC K@RTE pour la centralisation des logs à choisir par le projet et K@RTE, à créer par l'équipe K@RTE.

> Format: {{ENV}}-APP_{{NNI}}_centralisationLogs_01. Ex: DEV-APP_1033_centralisationLogs_01 * Dans la section 'output.kafka': * username: '' * password: '' * et resp. les user et mdp pour l'accès à K@RTE * : Client ID pour l'accès à K@RTE

Adresse de contact K@RTE: RTE-DSIT-DATA-DEMANDES-FLUX

Connexion à MARVEL

Adresse de contact MARVEL: design-osa-supervision.dsit@rte-france.com

Demandes: * Envoi de formulaire de demande de raccordement: Offre de Service MARVEL - informations détaillées - template.docx * Création de compte sur MARVEL

> **Informations utiles**
>
> - Dans le formulaire, bien prendre en compte que la partie « liste des flux », permet à l'équipe MARVEL de créer un index par flux identifié.
> - Chaque index est indépendant. Pour de la recherche croisée, il vaut mieux déverser la totalité des logs dans un seul index pour tous les microservices d'un environnement.
> - Pas de structure particulière attendue dans la formation des logs pour la consommation par MARVEL (filebeat s'en charge déjà).
> - Pas de prise en charge de la donnée par MARVEL, chaque projet est responsable de sa donnée et du détail qu'il trace dans ses logs. /!\ Cela induit également que la gestion de la thématique RGPD doit être portée par les projets. L'anonymisation doit être faite en amont.
> - MARVEL a 1 environnement par plateforme, l'OPF est disponible et la PREPROD et PROD. Donc tous les environnements des projets en OPF (Dev, Recette, …) doivent pointer sur le même environnement MARVEL.
> - Chaque projet est indépendant pour son dashboard, néanmoins, l'espace de stockage des logs est commun à tous : environ 1,5 GO prévu/jour avec une rétention de 60 jours. La rétention est paramétrable, les 60 jours sont par défaut.
> - Il n'existe à ce jour pas de documentation particulière sur la construction du dashboard Kibana. Aide possible mais limitée de l'équipe MARVEL au besoin. Dans l'idée chaque projet monte en compétences sur l'outils pour créer son dashboard.
> - Peut-être la possibilité d'importer/exporter un dashboard HypeEOD dans un dashboard MARVEL (à vérifier)
> - Possibilité de dupliquer les dashboard en fonction des environnements de chaque projet (Dev, recette, …) sur MARVEL en utilisant la fonction import/export.

Vérifications finales

Se connecter à MARVEL et vérifier que tous les logs de l'application atterrissent bien dans l'index ELK

Guide de déploiement Startix avec Minikube

Table des matières

1. Vue d'ensemble
2. Prérequis
3. Installation de Minikube
4. Déploiement automatisé avec Makefile ⭐ **Recommandé**
5. Configuration du projet
6. Déploiement de l'application
7. Accès à l'application
8. Commandes utiles
9. Structure du dossier de déploiement
10. Résolution des problèmes
11. Nettoyage

Vue d'ensemble

Ce guide explique comment déployer l'application Startix localement avec Minikube **avec le proxy RTE**.

🚀 **Déploiement rapide** : Utilisez `make deploy-minikube` pour un déploiement automatisé en une seule commande (voir section 4).

Architecture locale

- **Backend** : Spring Boot avec PostgreSQL
- **Frontend** : Angular

- **Service** : NodePort (local), ClusterIP (PostgreSQL)
- **Images Docker** : `startix-back-local:latest` et `startix-front-local:latest`, construites localement
- **Image de base** : `inca.rte-france.com/rtsi/openjdk:21-alpine3.19`

Prérequis

- Linux (Ubuntu 18.04+)
- Docker 20.10+
- Java 21
- Maven 3.9.9
- `kubectl` et `minikube` installés dans `~/.local/bin`
- Accès à l'image OpenJDK INCA en local (prétéléchargée si besoin)

> **Note** : Pour plus d'informations sur l'accès au cluster Kubernetes, consultez la section Accès au cluster Kubernetes k8s avec Kubectl de la documentation Kubernetes.

Installation de Minikube

Installation de Minikube en local

Voici les commandes utilisées pour installer minikube en local :

```
mkdir -p $HOME/bin
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
mv minikube-linux-amd64 $HOME/bin/minikube
chmod +x $HOME/bin/minikube
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

Étapes exactes (environnement proxy entreprise)

```
# 1. Réinitialiser complètement le cluster et les configs
minikube delete
rm -rf ~/.kube
rm -rf ~/.minikube

# 2. Exporter les variables proxy AVANT le start
export HTTP_PROXY=http://nni:mot_de_passe@163.104.40.34:3128
export HTTPS_PROXY=http://nni:mot_de_passe@163.104.40.34:3128

# 3. Démarrer Minikube avec les options spécifiques
minikube start \
  --kubernetes-version=stable \
  --extra-config=apiserver.enable-admission-plugins="" \
  --extra-config=kubelet.fail-swap-on=false

# 4. Récupérer l'IP de Minikube et configurer NO_PROXY
MINIKUBE_IP=$(minikube ip)
export NO_PROXY=${NO_PROXY},${MINIKUBE_IP}

# 5. Vérifier l'accès API (doit retourner 'yes')
minikube kubectl -- auth can-i create namespace
```

Déploiement automatisé avec Makefile

> 💡 **Méthode recommandée** : Une fois Minikube installé et démarré, utilisez le Makefile pour automatiser tout le processus.

Déploiement en une seule commande

```
# Depuis la racine du projet (startix/)
make deploy-minikube
```

Cette commande unique **automatise toutes les étapes** suivantes : - Configuration du proxy RTE - Compilation du backend (Maven) et du frontend (npm) - Construction des images Docker backend et frontend - Suppression des anciens pods et images - Chargement des nouvelles images dans Minikube - Déploiement des ressources Kubernetes (PostgreSQL, Backend, Frontend)

Accès à l'application

Exposer les services startix-front et startix-back via port-forward:

```
# Depuis la racine du projet (startix/)
make port-forward
```

L'application sera exposée : - **Frontend** : http://localhost:9000 - **Backend** : http://localhost:8080
Appuyez sur `Ctrl+C` pour arrêter le port-forwarding.

Autres commandes utiles

```
make help          # Affiche toutes les commandes disponibles
make status        # Vérifie l'état des pods
make logs          # Affiche les logs du backend
make port-forward  # Relance le port-forward si nécessaire
make clean         # Supprime le déploiement
```

> 📝 **Note** : Les sections suivantes détaillent les étapes manuelles pour comprendre ce que fait le Makefile en interne.

Configuration du projet

> ℹ️ **Note** : Cette section décrit les étapes manuelles. Si vous utilisez `make deploy-minikube`, ces étapes sont automatisées.

Compilation du backend

```
cd backend
./mvnw clean package -DskipTests
```

Construction de l'image Docker + chargement dans Minikube

```
unset DOCKER_TLS_VERIFY DOCKER_HOST DOCKER_CERT_PATH MINIKUBE_ACTIVE_DOCKERD
docker build -t startix-back-local:latest .
minikube image load startix-back-local:latest
```

Compilation du frontend

```
cd frontend
npm run build
```

Construction de l'image Docker + chargement dans Minikube

```
docker build -t startix-front-local:latest .
minikube image load startix-front-local:latest
```

Déploiement de l'application

> ℹ️ **Note** : Cette section décrit le déploiement manuel. Pour un déploiement automatisé, utilisez `make deploy-minikube`.

Déploiement manuel

```
# 1. Créer le namespace
minikube kubectl -- create namespace ns-startix-local --validate=false

# 2. Appliquer les manifestes Kustomize
# ⚠️ Exécuter depuis la racine du projet (startix/)
minikube kubectl -- apply -k deployment/k8s/ns-startix-local --validate=false

# 3. Attendre que le pod soit prêt
minikube kubectl -- wait --for=condition=Ready pod -l app=startix-back -n ns-startix-local --timeout=300s
minikube kubectl -- wait --for=condition=Ready pod -l app=startix-front -n ns-startix-local --timeout=300s
```

Accès à l'application

Accès depuis un navigateur local

1- Via l'IP Minikube

Remarque: Avec l'intégration OAuth2/OIDC GAIA en local, le serveur d'authentification (PingFederate) ne reconnaît pas l'origine de ta requête (IP Minikube), ce qui provoque des erreurs de **redirection invalide** et des problèmes de **CORS**. Voici comment corriger ça proprement : - Demander la déclaration de l'URL Minikube dans GAIA PingFederate comme redirect URI avec l'autorisation CORS en passant par une demande COSMOS. - Utiliser `kubectl port-forward` pour contourner l'IP Minikube. (Passer directement à la section suivante)

Lancer cette commande qui retourne l'URL complète (IP Minikube avec le port):

```
minikube service startix-back --url -n ns-startix-local
minikube service startix-front --url -n ns-startix-local
```

Pour la partie Backend:

- Ouvrir dans le navigateur :

```
## Exemple: Startix-back ##
http://<url_retournée>/bonjour
http://192.168.49.2:32470/bonjour
```

NB: Il est possible que la page ne réponde pas immédiatement : **Spring Boot peut prendre 30 à 40 secondes à démarrer**.

Pour la partie Frontend:

- Vérifier la configuration dans la configmap.yaml et modifier la variable d'environnement suivant: `OIDC_REDIRECT_URI: "<url_retournée>/"`
- Recharger la configuration (si modification):

```
minikube kubectl -- rollout restart deployment/startix-front-deployment -n ns-startix-local
```

- Ouvrir ensuite dans le navigateur :

```
## Exemple: Startix-front ##
http://<url_retournée>/
http://192.168.49.2:32470/
```

2- Via le port-forward

```
minikube kubectl -- port-forward svc/startix-front 9000:80 -n ns-startix-local
```

- Vérifier la varible d'environnement OIDC_REDIRECT_URI: "http://localhost:9000/" configuration dans la configmap.yaml
- Recharger la configuration (si modification):

```
minikube kubectl -- rollout restart deployment/startix-front-deployment -n ns-startix-local
```

- Puis accède à via http://localhost:9000

Vérification interne (debug réseau dans le cluster)

```
minikube kubectl -- exec -n ns-startix-local deployment/startix-back-deployment -- wget -qO- http://localhost:
```

Commandes utiles

```
# Arrêter Minikube
minikube stop

# Recompiler + redéployer
cd backend
./mvnw clean package -DskipTests
unset DOCKER_TLS_VERIFY DOCKER_HOST DOCKER_CERT_PATH MINIKUBE_ACTIVE_DOCKERD
docker build -t startix-local:latest .
minikube image load startix-local:latest
minikube kubectl -- rollout restart deployment/startix-deployment -n ns-startix-local
```

```
# Logs + shell
minikube kubectl -- logs -f deployment/startix-deployment -n ns-startix-local
minikube kubectl -- exec -it deployment/startix-deployment -n ns-startix-local -- /bin/sh

# Lister les pods dans un namespace
minikube kubectl -- get pods -n ns-startix-local

# Lister les services dans un namespace
minikube kubectl -- get svc -n ns-startix-local

# Voir les IP internes des services/pods
minikube kubectl -- get svc -o wide -n ns-startix-local
minikube kubectl -- get pods -o wide -n ns-startix-local

# Voir les logs d'un pod spécifique
minikube kubectl -- logs <nom-du-pod> -n ns-startix-local
```

Structure du dossier de déploiement

Le dossier `deployment/k8s/ns-startix-local/` contient le kustomize :

- `kustomization.yaml` : indique à Kustomize quelles ressources utiliser (avec la possibilité d'ajouter des patchs à appliquer).

Le dossier `deployment/k8s/components/minikube-startix-back/` contient les fichiers nécessaires au déploiement de la partie **Backend** (deployment.yaml, service.yaml et kustomization.yaml) :

- `deployment.yaml` : décrit un **Deployment** qui crée et maintient un pod exécutant l'application.
- `service.yaml` : expose le pod via un **Service** Kubernetes, accessible par NodePort.
- `kustomization.yaml` : indique à Kustomize quelles ressources utiliser.

Le dossier `deployment/k8s/components/minikube-startix-front/` contient les fichiers nécessaires au déploiement de la partie **Frontend** (configmap.yaml, deployment.yaml, service.yaml et kustomization.yaml) :

- `configmap.yaml` : décrit un **Configmap** qui contient les variables d'environnement de l'application Frontend.
- `deployment.yaml` : décrit un **Deployment** qui crée et maintient un pod exécutant l'application.
- `service.yaml` : expose le pod via un **Service** Kubernetes, accessible par NodePort.
- `kustomization.yaml` : indique à Kustomize quelles ressources utiliser.

Pourquoi cette structure ?

Kustomize permet de **séparer la configuration générique (production, CI)** des **configurations locales spécifiques**. Cela évite d'éditer les fichiers YAML de base.

Pour chaque application, il est recommandé d'avoir **au minimum**:

Élément	Rôle
`Deployment`	Crée et maintient un ou plusieurs pods basés sur votre image Docker.
`Service`	Expose votre pod pour y accéder (par NodePort, ClusterIP, etc).
`Dockerfile`	Décrit comment construire l'image Docker `startix-local`.
`Kustomization`	Assemble le tout avec des patches si besoin (environnement local).

Résolution des problèmes

Problèmes courants

- **ErrImagePull** : Reconstruire l'image locale avec `docker build` puis `minikube image load`
- **403 Forbidden** : Réinitialiser avec `minikube delete` **+** `rm -rf ~/.kube ~/.minikube`
- **Accès bloqué** : Patienter après le déploiement ou tester avec `kubectl exec ... wget`
- **Erreur chemin Kustomize** : Vérifier que `deployment/k8s/ns-startix-local` est accessible depuis le répertoire courant
- **Proxy error** : Configurer `export NO_PROXY=*.rte-france.com,localhost,127.0.0.1,192.168.49.0/24,$(minikube ip)`

Solution rapide

En cas de problème avec le déploiement manuel, utilisez le Makefile qui gère automatiquement : - La configuration du proxy - La suppression des anciens pods/images - Le rechargement des nouvelles images - L'exposition des services

```
make clean              # Nettoyer complètement
make deploy-minikube    # Redéployer proprement
```

Nettoyage

Avec Makefile (recommandé)

```
make clean    # Supprime le namespace et tous les pods
```

Manuel

```
minikube kubectl -- delete -k deployment/k8s/ns-startix-local --validate=false
minikube kubectl -- delete namespace ns-startix-local
minikube delete    # Supprime complètement Minikube (optionnel)
```

Important : Toujours utiliser `minikube kubectl --` dans cet environnement proxy.

Documentation PostgreSQL (INPUT RTE)

À propos

L'infrastructure INPUT a pour but de permettre l'hébergement mutualisé des bases de données PostgreSQL. Page GoPro INPUT

Contexte

Le projet Startix est basé sur une architecture Angular SPA en frontend + un microservice Spring Boot en backend exposant des API CRUD pour gérer des notes. On a choisi d'implémenter la base de données SQL Postgres (INPUT) pour sauvegarder les notes.

Choix de la base de données

RTE fournit une offre de service **PostgreSQL INPUT** et **Oracle EXADATA**.

Le choix de la base de données doit être fondé sur les besoins métier, techniques et opérationnels du projet. Voici quelques critères essentiels : - La nature des données (structurées ou flexibles), - La complexité des relations entre entités, - Les besoins en performance et scalabilité, - Les exigences de sécurité et conformité, - L'écosystème technologique utilisé (frameworks, outils BI, langages), - Le coût et le modèle de licence (open source ou propriétaire),

SQL ou NoSQL

- **Une base relationnelle (SQL)** est généralement recommandée pour des données bien structurées et des opérations transactionnelles fiables
- **Une base non relationnelle (NoSQL)** convient mieux aux données semi-structurées, aux volumes massifs ou aux architectures distribuées.

Cas d'usage typiques

Cas	Recommandation
ERP, CRM, finance, RH	SQL (PostgreSQL, Oracle)
Logs, IoT, analytics, réseaux sociaux	NoSQL (MongoDB, Cassandra)
Microservices REST avec JWT	SQL ou NoSQL selon la structure des données

Le choix doit donc refléter les contraintes fonctionnelles, les objectifs techniques et les perspectives d'évolution du projet.

Base	Licence	Coût	Support	Pourquoi
PostgreSQL	Open source	Gratuit	Communauté + support pro possible	Structuré, robuste, open source, parfait pour CRUD + microservices
Oracle (SQL)	Propriétaire	Payant	Support officiel	Environnement entreprise avec exigences fortes en sécurité, audit, SLA
MongoDB (NoSQL)	Open source / cloud	Gratuit ou payant	Support cloud	Stocker des notes comme documents JSON avec structure libre

Recommandation :

Actuellement, INPUT utilise la version PostgreSQL 14 : - Si la taille de la base est inférieure à 50 Go, utiliser INPUT. - Au-delà de 50 Go, utiliser Exadata.

Pour la future version d'INPUT basée sur PostgreSQL 17 : - La limite sera portée à 300 Go. - Au-delà de 300 Go, il faudra utiliser Exadata.

Note : La version PostgreSQL 17 est déjà disponible en OPF depuis 09/2025 et sera disponible en OPFi vers la fin de l'année 2025.

Tutoriel / Intégration Postgresql (INPUT)

L'infrastructure INPUT a pour but de permettre l'hébergement mutualisé des bases de données PostgreSQL.

Remarque : Input permet l'hébergement des bases de données de 50 go par défaut, si vous souhaitez disposer d'un base plus importante merci de prendre contact avec l'équipe pépite.

Elle fonctionne avec des version 10.x, 12.x et 14.x de PostgreSQL.

1. Dépendances pom.xml

- Ajouter ces dépendances dans le pom.xml
- JPA
- Postgresql
- Liquibase (gestion des versions de bases de données)
- TestContainer (Utilisation des conteneurs Docker éphémères pour les tests d'intégration)

```
<!-- Spring Data JPA -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate.orm</groupId>
    <artifactId>hibernate-core</artifactId>
</dependency>

<!-- PostgreSQL Driver -->
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>${liquibase.version}</version>
</dependency>

<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>postgresql</artifactId>
    <version>${testcontainers.version}</version>
    <scope>test</scope>
</dependency>
```

- Ajouter les versions dans les propriètes

```
<properties>
    <testcontainers.version>1.20.3</testcontainers.version>
    <liquibase.version>4.29.2</liquibase.version>
</properties>
```

Note: La verison SpringBoot utilisée dans ce projet est la v3.5.3

2. Properties de configuration

- application.yml: Contient la configuration spring necessaire pour la datasource, jpa, jdbc (etc...)

```
spring:
  data:
    jpa:
      repositories:
        bootstrap-mode: deferred
  datasource:
    driver-class-name: org.postgresql.Driver
    hikari:
      auto-commit: false
      poolName: Hikari
    type: com.zaxxer.hikari.HikariDataSource
    url: jdbc:postgresql://localhost:5432/startix
    username: startix
    password: ''
  jpa:
    hibernate:
      ddl-auto: none
      naming:
        implicit-strategy: org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
        physical-strategy: org.hibernate.boot.model.naming.CamelCaseToUnderscoresNamingStrategy
    open-in-view: false
    properties:
      hibernate:
        connection:
          provider_disables_autocommit: true
        generate_statistics: false
        jdbc:
          batch_size: 25
          time_zone: UTC
        order_inserts: true
        order_updates: true
        query:
          fail_on_pagination_over_collection_fetch: true
          in_clause_parameter_padding: true
```

Mettre à jour si besoin `l'url`, `username` et le `password` de la base de données.

- application-test.yml: Contient la configuration de la base de données Postgresql pour les Tests integrations en utilisant les TestsContainer.

```
spring:
  datasource:
    driver-class-name: org.testcontainers.jdbc.ContainerDatabaseDriver
    hikari:
      maximum-pool-size: 2
    url: jdbc:tc:postgresql:14.1:///startix?TC_TMPFS=/testtmpfs:rw
    username: startix
    password: ''
```

Externaliser les variables d'environnement SPRING_DATASOURCE dans les fichiers des variables d'environnement : - env.conf

```
## PERSISTANCE CONFIG ##
SPRING_DATASOURCE_HOST=localhost
SPRING_DATASOURCE_PORT=5432
SPRING_DATASOURCE_DBNAME=startix
SPRING_DATASOURCE_URL=jdbc:postgresql://${SPRING_DATASOURCE_HOST}:${SPRING_DATASOURCE_PORT}/${SPRING_DATASOURC
```

- private.conf

```
## PERSISTANCE CONFIG ##
SPRING_DATASOURCE_USERNAME=startix
SPRING_DATASOURCE_PASSWORD=
```

3. Ajouter la classe @Configuration SpringBoot

- DatabaseConfiguration.java

```
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.transaction.annotation.EnableTransactionManagement;
```

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(basePackages = { "com.rte.fr.startix" }, enableDefaultTransactions = false)
class DatabaseConfiguration { }
```

4. Ajouter le conteneur Docker Postgresql

postgresql.yml : Ce fichier est utilisé pour définir et lancer un service Docker PostgreSQL prêt à être utilisé pour le développement.

```
version: '3'
services:
  postgresql:
    image: postgres:14.1
    container_name: postgresql
    volumes:
      - postgresql-data:/var/lib/postgresql/data/
    environment:
      - POSTGRES_USER=startix
      - POSTGRES_PASSWORD=
      - POSTGRES_HOST_AUTH_METHOD=trust
    # If you want to expose these ports outside your dev PC,
    # remove the "127.0.0.1:" prefix
    ports:
      - '127.0.0.1:5432:5432'
volumes:
  postgresql-data:
```

Aller sous le dossier `backend` et lancer la base de données:

```
docker-compose -f src/main/docker/postgresql.yml up -d
```

Vérifier que le service postgresql est UP:

```
docker ps
```

5. Ajouter le conteneur PgAdmin

pgAdmin est un outil qui permet de gérer les bases de données PostgreSQL

pgadmin.yml : Permet de définir le service Docker PgAdmin

```
version: '3'

services:
  pgadmin:
    image: dpage/pgadmin4:8.3
    container_name: pgadmin
    expose:
      - 80
    ports:
      - 8088:80
    environment:
      - PGADMIN_DEFAULT_EMAIL=pgadmin@mail.com
      - PGADMIN_DEFAULT_PASSWORD=adminadmin
    volumes:
      - pgadmin-data:/var/lib/pgadmin
    restart: unless-stopped
volumes:
  pgadmin-data:
```

- Aller sous le dossier `backend` et lancer la base de données:

```
docker-compose -f src/main/docker/pgadmin.yml up -d
```

Le service PgAdmin se lance sur http://localhost:8088

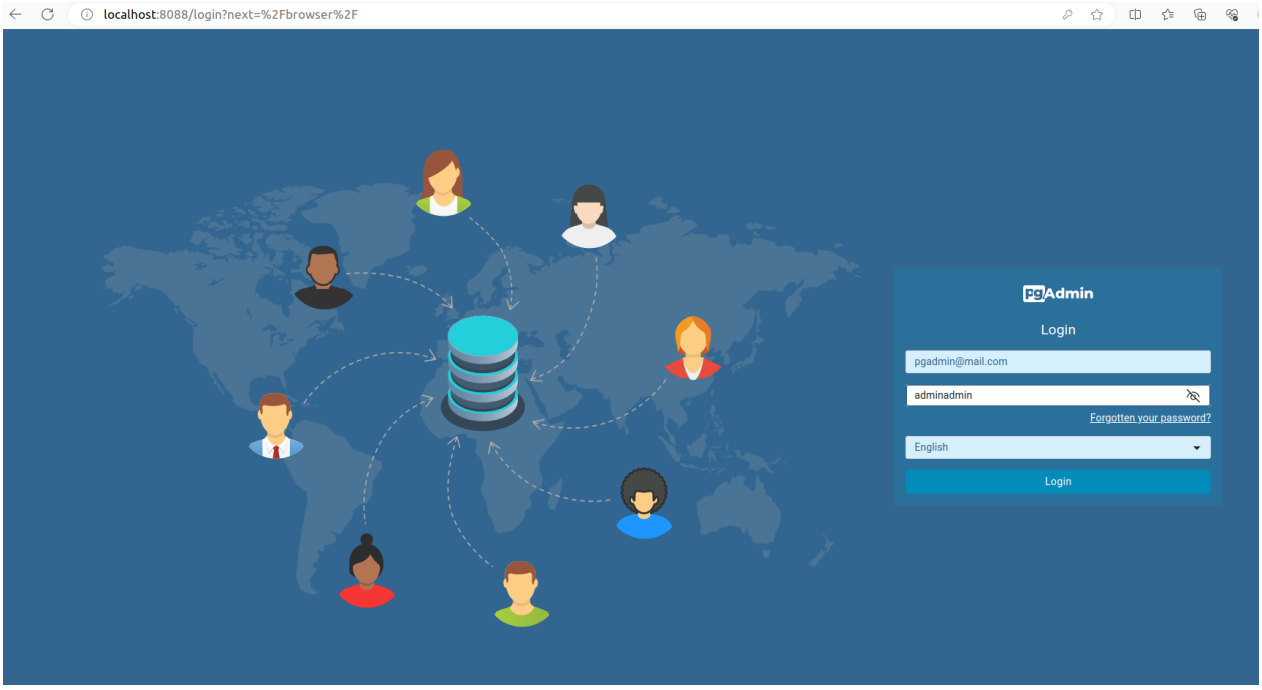
- Vérifier que les services postgresql et pgAdmin sont UP:

```
docker ps
```

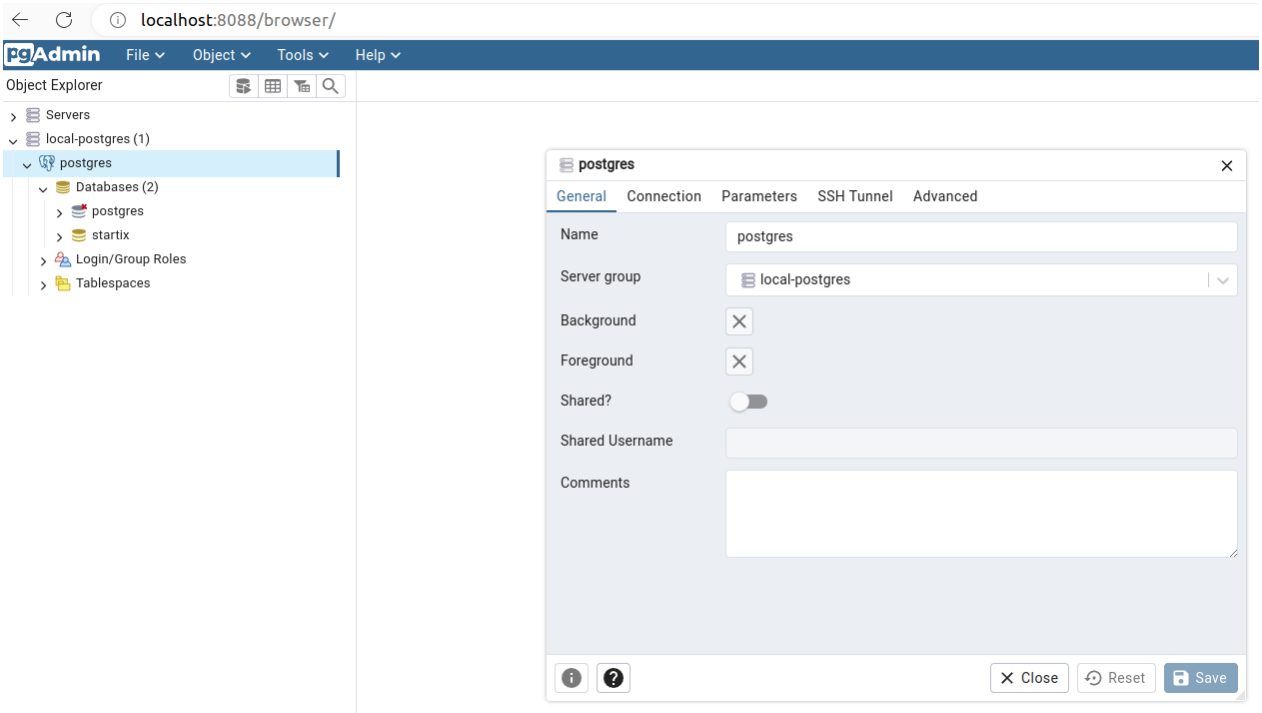
- Visualiser les logs des services:

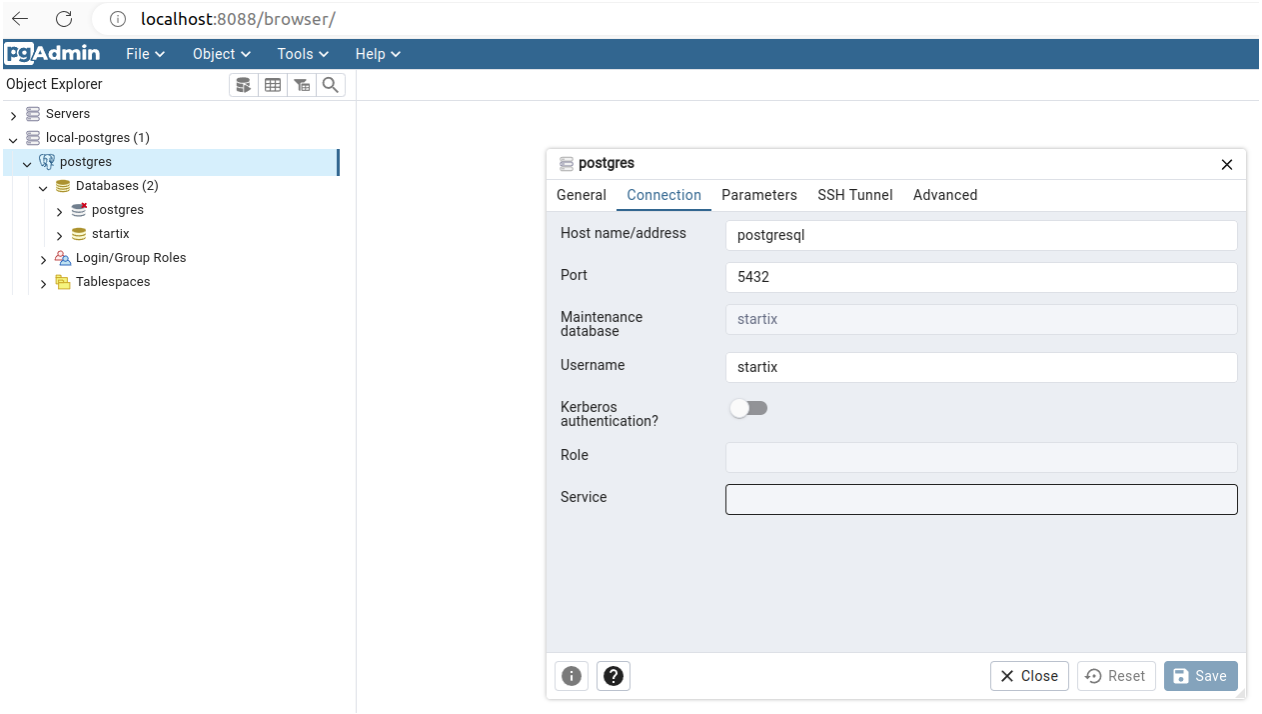
```
docker logs postgresql
docker logs pgadmin
```

- Se connecter à l'interface PgAdmin sur http://localhost:8088/login:
- email: pgadmin@mail.com
- mdp: adminadmin

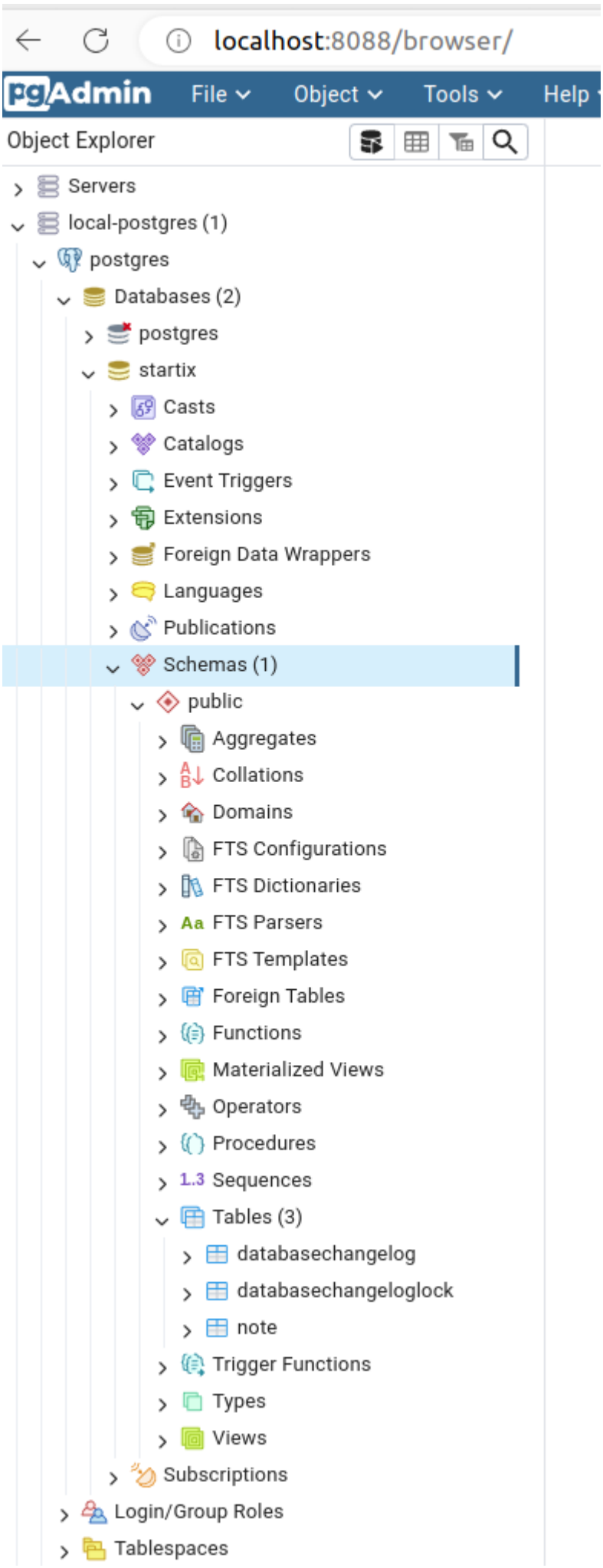


- Cliquer sur "Register/Server" dans un "Server Group" et ajouter les informations de connexion à la base de données Postgresql





- Paramètres de connexion:
- name: Postgres (Le nom choisi de la config)
- Host: postgresql (Le service postgresql qui tourne en local. Voir la section dédiée)
- Port: 5432 (postgresql port)
- Maintenance database: startix (le nom de la BD)
- Username: startix (l'utilisateur de la base)
- Trouver ensuite la base de données `startix` et les tables dans l'arborescence à gauche:


localhost:8088/browser/
pgAdmin
File
Object
Tools
Help
Object Explorer
Servers
local-postgres (1)
postgres
Databases (2)
postgres
startix
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (3)
databasechangelog
databasechangeloglock
note
Trigger Functions
Types
Views
Subscriptions
Login/Group Roles
Tablespaces

6. Ajouter la configuration Liquibase

Liquibase est un outil open-source de gestion des versions de bases de données. Il permet de suivre, versionner et automatiser les changements apportés à la structure d'une base de données.

Voici un exemple d'une configuration Liquibase qui permet d'initialiser la base de données et créer une table `note`

- Ajouter la configuration du Liquibase dans le fichier application.yml

```
spring:
  liquibase:
    change-log: classpath:config/liquibase/master.xml
```

- Ajouter le fichier master.xml sous `backend/src/main/resources/config/liquibase`

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
        http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.3.xsd">

  <property name="uuidType" value="uuid" dbms="h2, postgresql" />

  <include file="config/liquibase/changelog/init_note_schema.xml" relativeToChangelogFile="false"/>
</databaseChangeLog>
```

- Ajouter le fichier init_note_schema.xml sous `backend/src/main/resources/config/liquibase/changelog`

```
<databaseChangeLog
        xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
        http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-4.3.xsd">

  <changeSet id="1" author="startix">
    <createTable tableName="note">
      <column name="id" type="${uuidType}">
        <constraints nullable="false" primaryKey="true" />
      </column>

      <column name="name" type="varchar(255)">
        <constraints nullable="false" />
      </column>

      <column name="content" type="TEXT">
        <constraints nullable="false" />
      </column>

      <column name="created_by" type="VARCHAR(255)">
        <constraints nullable="false" />
      </column>
    </createTable>
  </changeSet>
</databaseChangeLog>
```

- Lancer le projet spring boot et vérifier la connexion à a base de données postgresql et la création de la table dans la base par Liquibase en utilisant le PgAdmin

```
mvn spring-boot:run
```

Administratif

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	INPUT		

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
		Plateformes OPF	- https://cosmos.rte-france.com/esc?id=sc_cat_item&sys_id=4882d35e35b4a11074401bd68eeb52e8 - https://gopro-collaboratif.rte-france.com/display/INP/OPF

D1. Demande d'une base de données PostgreSQL.

Cette demande permet la Création - Modification - Suppression d'une base de données PostgreSQL.

- Nom court : nom court de votre application
- NNA : obligatoire même pour les POC
- Département Porteur : département porteur de l'application
- Paramètres spécifiques : demande particulières sur la base, ex encodage …

A savoir avant de développer avec INPUT

Transactions non-ACID

Il faut savoir qu'avec INPUT, par défaut, les transactions PostgreSQL ne sont pas ACID, même en prod.

C'est un choix fait par l'équipe qui gère INPUT pour garantir un bon niveau de performances et de disponibilité. Ce n'est pas un problème en soi tant que les développeurs sont au courant. Cette page GoPro explique le sujet en détail et différentes solutions de contournement pour les projets : https://gopro-collaboratif.rte-france.com/pages/viewpage.action?pageId=527093566

Analyse Sonar par Merge Request

À propos

On souhaite pouvoir effectuer une analyse Sonar des Merge requests (MR) avant qu'elles ne soient intégrées dans la branche principale.

Au final, on a besoin que la CI se lance dans les cas suivants :

- Analyse complète :
- de la branche principale lorsque celle-ci varie (après intégration d'une MR).
- des branches de livraison (branches "release-v…").
- Analyse différentielle, pour les MR :
- lors de la création de la MR.
- lors de l'ajout de commits à la MR.
- lors de la modification des commits de la MR (cas du rebase de la branche).
- sur demande, par ajout d'un commentaire "build" à la MR.

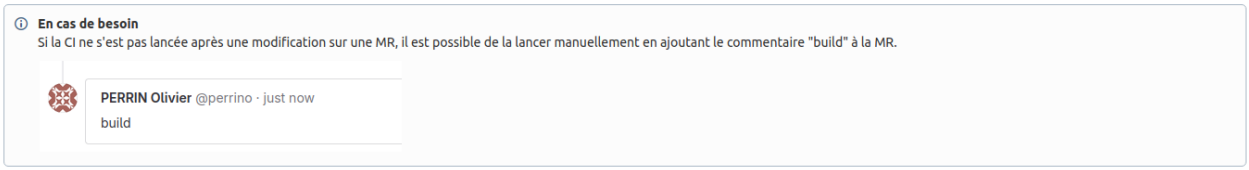
Prérequis

- Un Webhook sur Gitlab. Voir la section dédiée.
- Un Job Jenkins. Voir la section dédiée.
- Une étape Génération de rapport Sonar dans le Jenkinsfile.

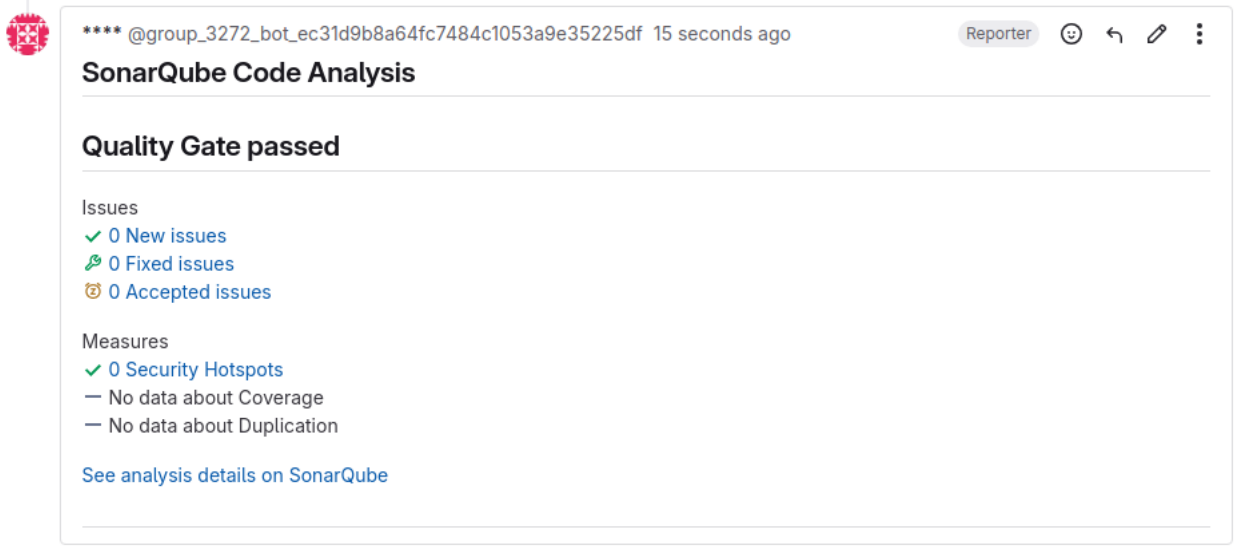
Workflow d'une MR

- Création de la branche avec les premiers commits et push de celle-ci sur le dépôt Gitlab.
- ❌ Pas de lancement de la CI : le nom de la branche n'est normalement ni "main", ni "release-v…".
- Création d'une MR (au statut Draft ou non)
- ✅ Lancement de la CI sur la MR
- Ajout de nouveaux commits à la MR
- ✅ Lancement de la CI sur la MR
- Modification de la description de la MR, ajout/suppression d'un tag
- ❌ Pas de lancement de la CI : le contenu de la branche n'a pas bougé.
- Approbation de la MR
- ❌ Pas de lancement de la CI : le contenu de la branche n'a pas bougé.
- Merge de la MR dans "main"
- ✅ Lancement de la CI sur "main"

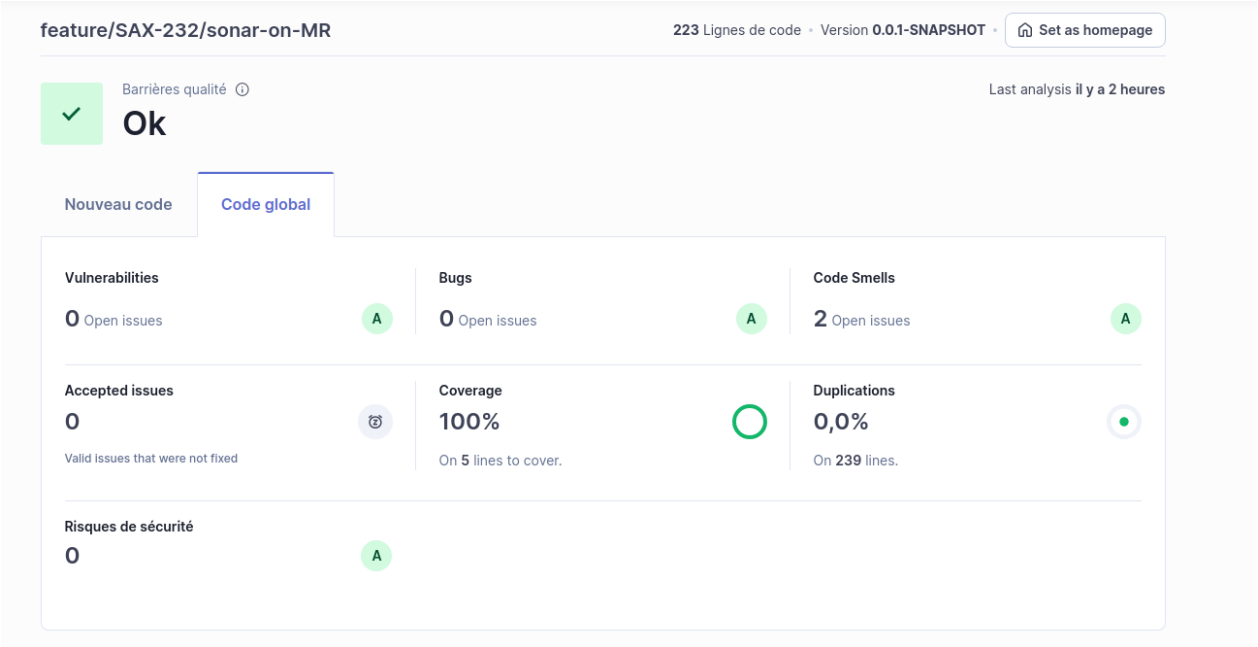
Note : dans le cas du changement de la branche cible d'une MR (celle dans laquelle la MR sera mergée), la CI ne se lance pas automatiquement. Il convient alors de la lancer manuellement (cf. encart ci-dessous).



À la fin de la CI, les rapports d'analyse Sonar des MR sont accessibles: - Depuis un commentaire ajouté automatiquement par Sonar à la MR. - Depuis Jenkins : le lien apparaît dans le résumé de la construction, dans la partie gauche de la fenêtre :



• Depuis Sonar, en sélectionnant la MR dans le menu de sélection des branches / MR:



Tutoriel / Intégration

1. Configuration côté Gitlab

Dans "devin-source" (gitlab), accéder au projet startix, puis à sa page de configuration "Settings" > "Webhooks" et éditer les détails du crochet vers devin-ic (jenkins).

Sélectionner les triggers suivants : - Push events - Regular expression : `^(main|release-v)` - Comments - Confidential comments - Merge request events

La regex permet de limiter l'envoi des notifications de push qu'aux seules branches "main" et de livraison. Les options "Comments" et "Confidential comments" permettent de lancer l'exécution d'un pipeline à partir d'un commentaire de

MR (configuré dans les builds triggers de Jenkins). L'option "Merge request events" permet de lancer les analyses lors de modifications sur une branche faisant l'objet d'une MR.

Astuce: Dans la page de configuration du webhook, on peut voir en fin de page les évènements envoyés vers Jenkins. Ceci peut aider à mieux comprendre pourquoi un job Jenkins a été lancé ou non.

2. Configuration côté Jenkins

Dans "devin-ic" (Jenkins), accéder au pipeline startix, puis à sa page de configuration "Configurer". Aller ensuite dans la section "Build triggers".

Activer "Build when a change is pushed to Gitlab (...)" et sélectionner les options suivantes :

- Push events
- Opened Merge Request events
- Build only if new commits were pushed to Merge Request
- ~~Accepted Merge Request events~~
- Rebuild open Merge Requests :
- On push to source branch
- Comments
- Comment (regex) for triggering a build: `build`
- Options avancées :
- Enable [ci-skip]
- ~~Ignore WIP Merge Requests~~
- Set build description to build cause (eg. Merge request or Git Push)
- Cancel pending merge request builds on update
- Allow all branches to trigger this job
- Secret token (même token que dans GitLab)

3. Configuration côté SonarQube server

Une configuration est nécessaire côté SonarQube server pour que celui-ci envoie les rapports d'analyse sur les MR dans Gitlab.

Cette configuration n'est pas à la main des projets et il faut la demander à l'équipe DevOps via un ticket JIRA.

Pour plus de détails, voir la section dédiée

4. Modification du pipeline

L'analyse Sonar par branche / MR s'effectue en utilisant la tâche "sonar" de la pipelines-lib de RTE.

Pour fonctionner, elle a besoin de 3 éléments : - la branche à analyser. - la branche dans laquelle doit être mergée la MR. - l'identifiant de la MR dans le projet Gitlab.

Ces éléments peuvent être récupérés de la façon suivante :

- En début de script :

```
/**
 * Main branch on Git, the one where sonar analyses are launched without parameters
 * See https://docs.sonarqube.org/latest/branches/overview/
 **/
String MAIN_BRANCH = 'main'

/** branch being build */
String currentBranch

/** target branch, when triggered on a MR */
String targetBranch

/** Id of the MR in the Gitlab project */
String mrKey = env.gitlabMergeRequestIid // 'Iid': id of the MR for the project ('Id' is for the whole Gitlab)
```

- Après l'appel à la tâche "gitCheckout()" :

```
currentBranch = gitResult['GIT_BRANCH']

// Target branch for sonar. Only present when building a MR. MAIN_BRANCH by default
targetBranch = gitResult['GIT_TARGET_BRANCH'] ?: MAIN_BRANCH
```

- L'appel à la tâche sonar est ensuite le suivant :

```
stageDevin ('🔬 Code Quality') {
    echo "🔬 Analyse sonar du code"
```

```
    if (!mrKey?.trim()) {
        echo "Sonar analysis on branch ${currentBranch}"
        dir('backend') {
            sonar {
                branch = currentBranch
                delegate.targetBranch = targetBranch
                version = buildVersion
                mavenVersion = MVN_VERSION
                jdk = CURRENT_JDK
            }
        }
    } else {
        echo "Sonar analysis on MR ${mrKey} from ${currentBranch} into ${targetBranch}"
        dir('backend') {
            sonar {
                gitlabReports = false // No gitlab reports (they are reported as comments on the last commit)
                pullRequestKey = mrKey
                pullRequestBase = targetBranch
                pullRequestBranch = currentBranch
                jdk = CURRENT_JDK
                mavenVersion = MVN_VERSION
                skipQualityGate = false // By default, the quality gate is not checked
            }
        }
    }
}
```

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	SonarQube	DevOps	https://gopro-tickets.rte-france.com/browse/DEVMCO-7923

D1. Configuration côté SonarQube server

Demander à l'équipe DevOps via un ticket de support JIRA DEVMCO-7923 :

```
Bonjour,

Nous souhaiterions bénéficier des rapports Sonar dans les merge requests GitLab des projets suivants :

- startix (id Gitlab : 9730)

Nous allons activer l'analyse par MR (modification des jobs Jenkins à venir), mais il faudrait également que v

La documentation de la configuration est disponible à cette adresse.
(Pour information, cette configuration a déjà été effectuée pour le projet powsybl-rte-core).


Pourriez-vous effectuer cette configuration ?

Bien cordialement,
Firas.
```

Note : l'identifiant du projet GitLab peut être trouvé depuis GitLab, dans la page "Settings > General" du projet.

Références

- https://devin-source.rte-france.com/devin/pipelines-lib
- https://plugins.jenkins.io/gitlab-plugin/
- https://docs.sonarsource.com/sonarqube/9.9/analyzing-source-code/pull-request-analysis/

Génération d'un rapport SonarQube à destination du CORS'N

À propos

A partir du 1er Novembre 2024, les critères de qualité du code évalués par Sonarqube deviennent prescriptifs dans le cadre d'une Mise en Production pour l'ensemble des nouveaux projets et des projets en cours (sauf dérogation). De ce

fait, l'équipe DevOps vous propose une méthode afin d'exporter au format pdf le résultat de ces analyses afin de pouvoir les transmettre au CORSN lors de vos demandes de créneaux de MEP.

Pour plus d'information sur le fonctionnement interne de la génération de ce rapport, vous pouvez consulter le fichier readme.md du projet Algoleague, qui est le créateur et porteur de cette méthode : AlgoLeague / 📄 RTE Sonar reports · GitLab

Génération du rapport

Une nouvelle fonction `sonarPrescriptionReport` a été implémentée dans nos pipelines-lib, permettant de générer ce rapport. La pratique que nous recommandons est la suivante :

Configuration dans le Jenkinsfile

La génération de rapports SonarQube utilise **deux Jenkinsfiles distincts** avec des responsabilités séparées :

1. Jenkinsfile principal (ex: `backend/Jenkinsfile`)

Responsabilité : Analyse du code et envoi vers SonarQube

```
stageDevin ('🔬 Code Quality') {
    echo "🔬 Analyse sonar du code"
    dir('backend') {
        sonar {
            branch = currentBranch
            delegate.targetBranch = targetBranch
            version = buildVersion
        }
    }
}
```

2. Jenkinsfile de rapport (ex: `ic/Jenkinsfile`)

Responsabilité : Génération du rapport PDF uniquement (PAS de rebuild)

```
stageDevin ('📄 Rapport SonarQube') {
    echo "📄 Génération du rapport SonarQube"

    sonarPrescriptionReport {
        applicationDescriptionPath = "ic/application_description.yml"
        reportPath = "VotreProjet_Report.pdf"
        archiveReport = true
    }
}
```

💡 **Avantage de cette séparation :** - ✅ Pas de rebuild inutile pour générer le rapport - ✅ Pipeline de rapport léger et rapide - ✅ Analyse SonarQube et génération de rapport découplées

Configuration du fichier application_description.yml

Emplacement dans un projet réel

Pour votre projet (non-Startix) :

```
votre-projet/
├── src/main/java/              # Code source
├── src/test/java/              # Tests
├── ic/                         # ⭐ Répertoire IC à créer
│   ├── application_description.yml  # ⭐ Configuration rapport
│   └── Jenkinsfile            # ⭐ Pipeline rapport (optionnel)
├── Jenkinsfile                # Pipeline principal avec analyse SonarQube
└── pom.xml
```

Étapes pour votre projet : 1. **Créez un répertoire `ic/`** à la racine de votre projet 2. **Placez-y le fichier `application_description.yml`** 3. **Adaptez le chemin** dans votre configuration : `applicationDescriptionPath = "ic/application_description.yml"`

Configuration du fichier application_description.yml

- Vous aurez besoin d'un fichier application_description.yml pour générer votre rapport. Ce fichier contient les informations sur les modules et les branches à analyser.

Emplacement du fichier : - **Pour Startix :** Le fichier est placé dans le répertoire `ic/application_description.yml` à la racine du projet - **Pour un projet réel :** Créez un répertoire `ic/` à la racine de votre projet et placez-y le fichier `application_description.yml`

Structure du projet recommandée :

```
votre-projet/
├── src/
├── ic/
│   ├── application_description.yml
│   └── Jenkinsfile (pour les rapports)
├── Jenkinsfile (pipeline principal)
└── pom.xml
```

Un exemple issu de Cetautomatix vous est présenté ci-dessous, si vous souhaitez un template plus général vous pourrez en trouver un ici : [AlgoLeague / Prescription Report](#).

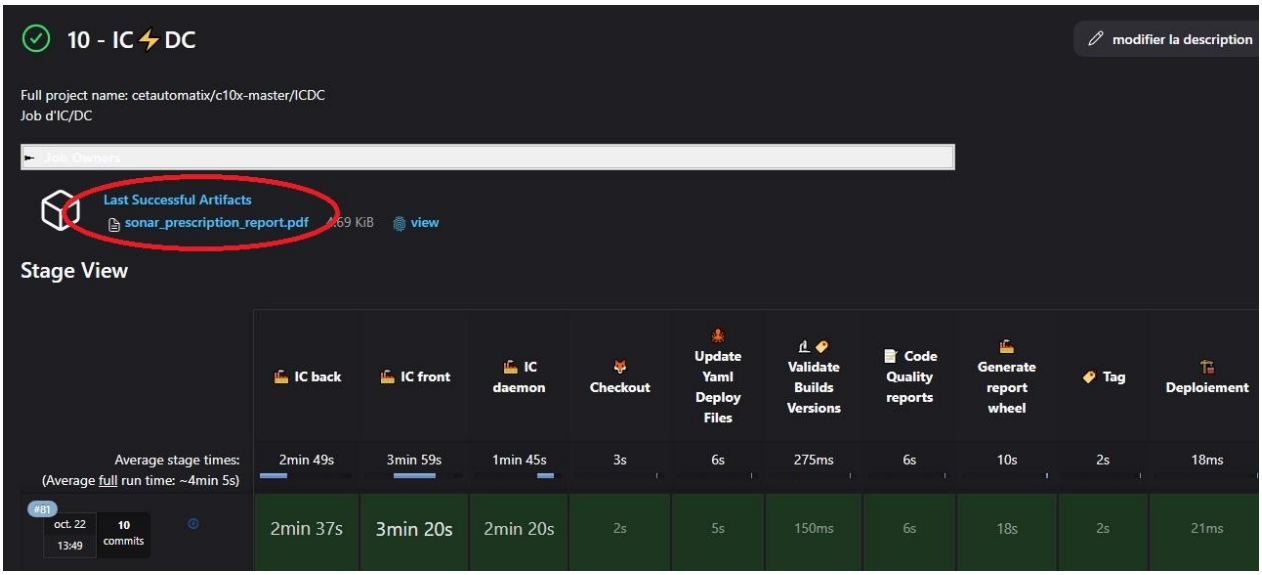
application_description.yml 596 B

```yaml
application:
  name: cetautomatix
  version: 4.0.1
  modules:
    - name: cetautomatix-back
      sonar_config: DevIn
      project_key: com.rte_france.cetautomatix:cetautomatix-back
      branch: test/sonar-report
      type: backend
    - name: cetautomatix-front
      sonar_config: DevIn
      project_key: com.rte_france.cetautomatix:cetautomatix-front
      branch: test/sonar-report
      type: frontend
    - name: cetautomatix-daemon
      sonar_config: DevIn
      project_key: com.rte_france.cetautomatix:cetautomatix-daemon
      branch: new-branch
      type: other
```

Configuration importante : Pour les projets qui sont sur le Gitlab de RTE, il faut mettre `sonar_config: DevIn`.

Récupération du rapport

Le rapport ainsi généré par le job Jenkins est récupérable en tant qu'artefact téléchargeable au format pdf.



Contacts

Pour toute question sur l'intégration de cette fonctionnalité dans vos pipelines DevOps, veuillez nous contacter par mail sur : rte-dsit-equipe-devops@rte-france.com

Pour toute question concernant les fonctionnalités propres à cette fonctionnalité, vous pouvez contacter : rte-dsit-prescriptions-techniques@rte-france.com

À propos

L'intégration SonarQube dans le projet Startix permet d'effectuer une analyse statique du code source pour détecter les bugs potentiels, vulnérabilités de sécurité et code smells. Cette intégration s'appuie sur plusieurs composants Maven et une configuration spécifique.

Tutoriel / Intégration SonarQube

Pré-requis

- Avoir un token pour se connecter à Sonar
- Avoir une CI sur Jenkins

1. Intégration de SonarQube sur le backend (avec Maven)

1.a. Utilisation de Sonar pour les projets Maven

Nous allons maintenant voir comment effectuer l'analyse SonarQube du code backend et envoyer les résultats vers l'instance SonarQube de RTE.

Utilisation : - Intégré automatiquement dans le pipeline Jenkins - Peut être exécuté manuellement avec : `mvn sonar:sonar` - Utilise les propriétés définies dans `sonar-project.properties` et dans `pom.xml`

Configuration RTE : - Il faut générer un token Sonar pour pouvoir se connecter depuis la CI - Les critères qualité sont définis au niveau de l'organisation RTE dans SonarQube. Les projets n'ont donc pas à définir des seuils de qualité.

1.b. Récupérer la couverture de code

Sonar a besoin d'un outil externe pour mesurer la couverture de code.Le RSSI impose d'utiliser JaCoCo pour les projets Java.

Le plugin JaCoCo génère les rapports de couverture de code nécessaires pour SonarQube.

Fonctionnalités : - Génération automatique des rapports de couverture pour les tests unitaires et d'intégration - Fusion des rapports pour une vue globale de la couverture - Export au format XML compatible SonarQube

Configurations possibles pour les projets : - Exclusions de packages/classes via `<excludes>` - Seuils de couverture minimale via `<rules>` - Formats de rapport supplémentaires (HTML, CSV)

2. Intégration de Sonar pour le frontend

2.a. Utilisation de Sonar pour les projets frontend

Nous allons maintenant voir comment effectuer l'analyse SonarQube du code front et envoyer les résultats vers l'instance SonarQube de RTE.

Utilisation : - Intégré automatiquement dans le pipeline Jenkins - Peut être exécuté manuellement avec : `mvn test sonar:sonar` - Utilise les propriétés définies dans `sonar-project.properties`

Configuration RTE : - Il faut posséder un token Sonar pour pouvoir se connecter depuis la CI. Ce token doit être généré par l'équipe DevOps lors de la création de l'espace Jenkins. Si votre projet n'a pas de token, il faut se rapprocher de l'équipe DevOps. - Les critères qualité sont définis au niveau de l'organisation RTE dans SonarQube. Les projets n'ont donc pas à définir des seuils de qualité.

2.b. Récupérer la couverture de code

Pour générer le rapport de couverture de tests pour la partie frontend, il faut lancer la commande `npm run test:coverage` dans le dossier `frontend`.

Les résultats sont stockés dans `target/test-results/`

3. Fichier sonar-project.properties

Ce fichier contient la configuration de base pour l'analyse SonarQube du projet Startix. Il définit les paramètres essentiels pour l'intégration avec l'instance SonarQube de RTE.

Propriétés configurables par les projets : - `sonar.projectKey` : Identifiant unique du projet dans SonarQube (cet attribut est récupéré automatiquement à partir du champ `groupId` du pom.xml pour les projets Maven) - `sonar.projectName` : Nom d'affichage du projet dans SonarQube (cet attribut est récupéré automatiquement à partir du champ `artifactId` du pom.xml pour les projets Maven) - `sonar.host.url` : URL de l'instance SonarQube (cet attribu est surchargé dans la pipeline Jenkins) - `sonar.token` : Token pour l'authentification (à commenter au moment de lancer la pipeline pour ne pas interférer avec les credentials de la CI)

4. Étape Code Quality dans le Jenkinsfile

L'étape "🔬 Code Quality" du pipeline Jenkins effectue une analyse statique du code source avec les objectifs suivants :

Fonctionnalités : - Détection des bugs potentiels, vulnérabilités de sécurité et code smells - Calcul de la couverture de code à partir des rapports de couverture de code - Génération d'un rapport détaillé dans l'instance SonarQube de RTE - Vérification du respect des critères qualité définis par RTE

Bonnes pratiques

Gestion des faux positifs

Principe : Les faux positifs et exclusions doivent être gérés directement dans l'interface SonarQube, pas dans le code du projet.

Raisons : - Maintient la propreté du code source - Centralise la gestion des règles dans SonarQube - Facilite la gouvernance et les audits - Permet une gestion collaborative des exceptions

Configuration pour l'environnement RTE

Production : - Modifier `sonar.host.url` pour pointer vers l'instance SonarQube de RTE - Configurer les credentials via des variables d'environnement sécurisées - Adapter `sonar.projectKey` selon les conventions de nommage RTE

Développement local : - Garder la configuration localhost pour les tests en local - Documenter la procédure de configuration pour les développeurs

Liens utiles

- Documentation SonarQube for IDE : Installation et configuration de l'extension IDE
- Génération de rapports SonarQube : Génération de rapports PDF pour le CORS'N
- Instance SonarQube RTE : Interface web de l'instance RTE

Tutoriel / Intégration SonarQube IDE

À propos

SonarQube pour IDE (anciennement SonarLint) est un outil d'analyse statique gratuit qui améliore la qualité et la sécurité de votre code. Cet outil permet d'analyser votre code dès sa rédaction, identifiant automatiquement les problèmes de qualité et de sécurité en temps réel, même avec du code généré par IA.

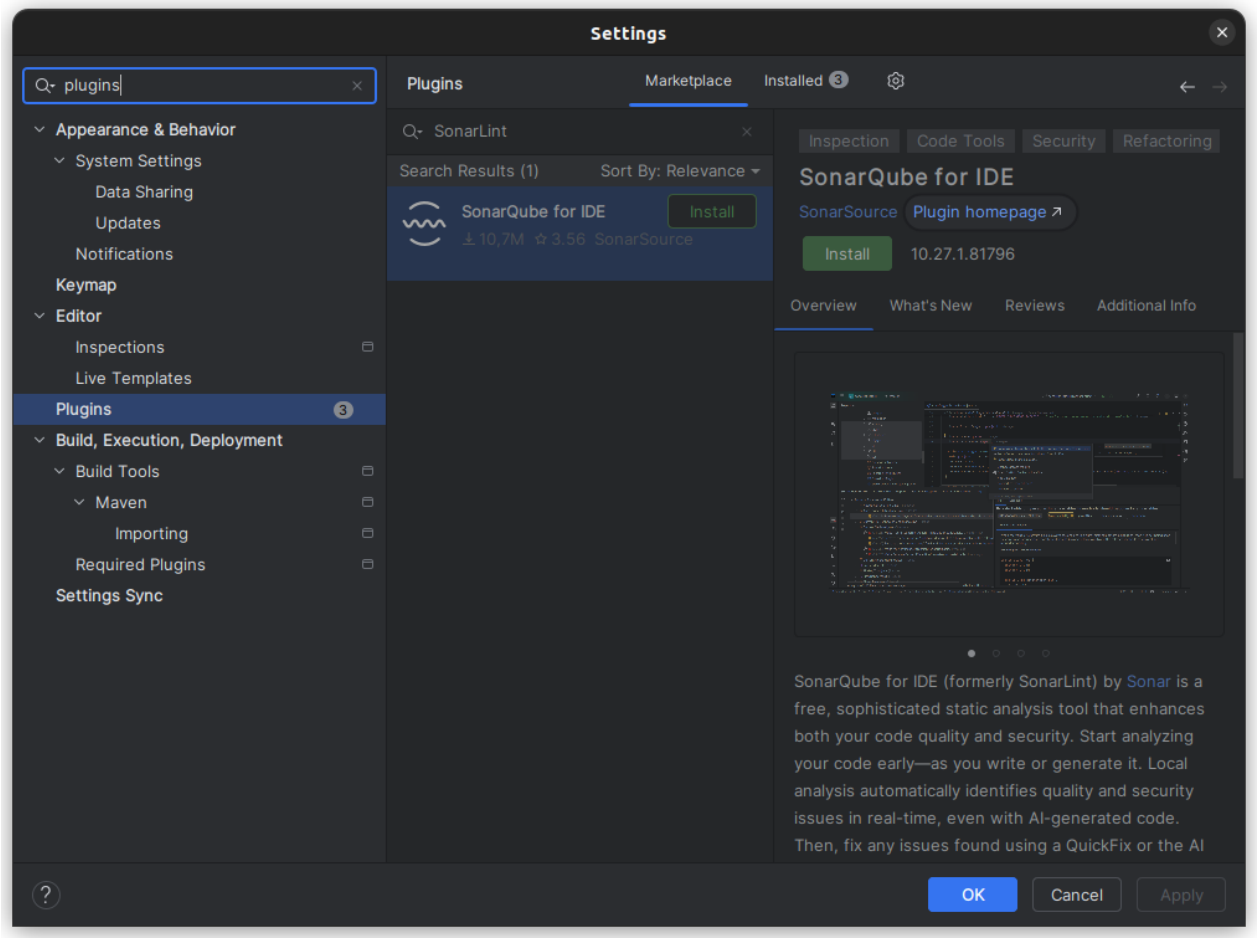
Cette documentation explique comment installer l'extension SonarQube for IDE et la connecter au serveur SonarQube de RTE.

Prérequis

- Un IDE compatible (IntelliJ IDEA, WebStorm, PyCharm, etc.)
- Un accès au serveur SonarQube de RTE (https://devin-qualite.rte-france.com/)

Installation de l'extension

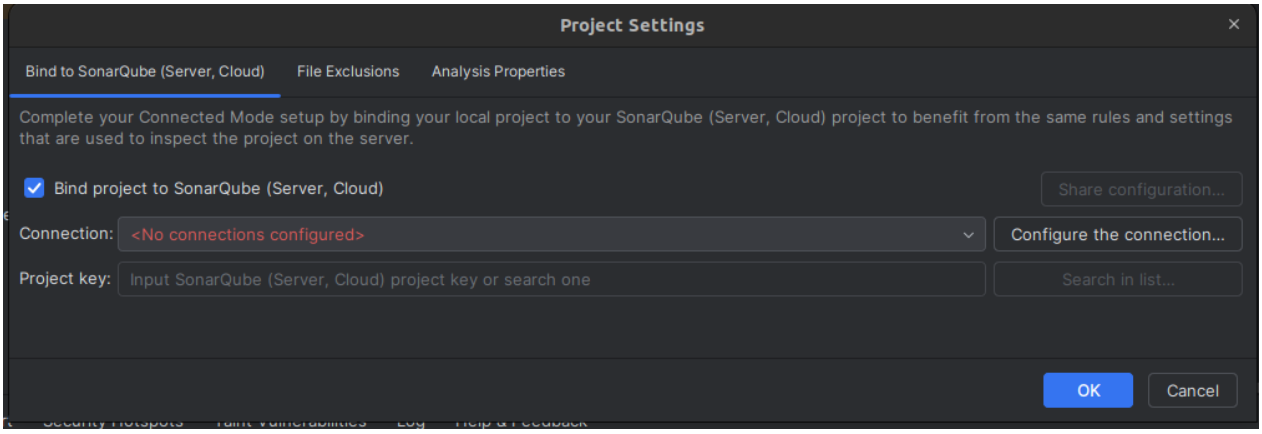
1. Ouvrez les paramètres de votre IDE
2. Accédez à la section "Plugins" dans le menu de gauche
3. Recherchez "SonarQube" ou "SonarLint" dans la barre de recherche du marketplace
4. Cliquez sur "Install" pour SonarQube for IDE



Configuration de l'extension

Étape 1 : Configuration de base

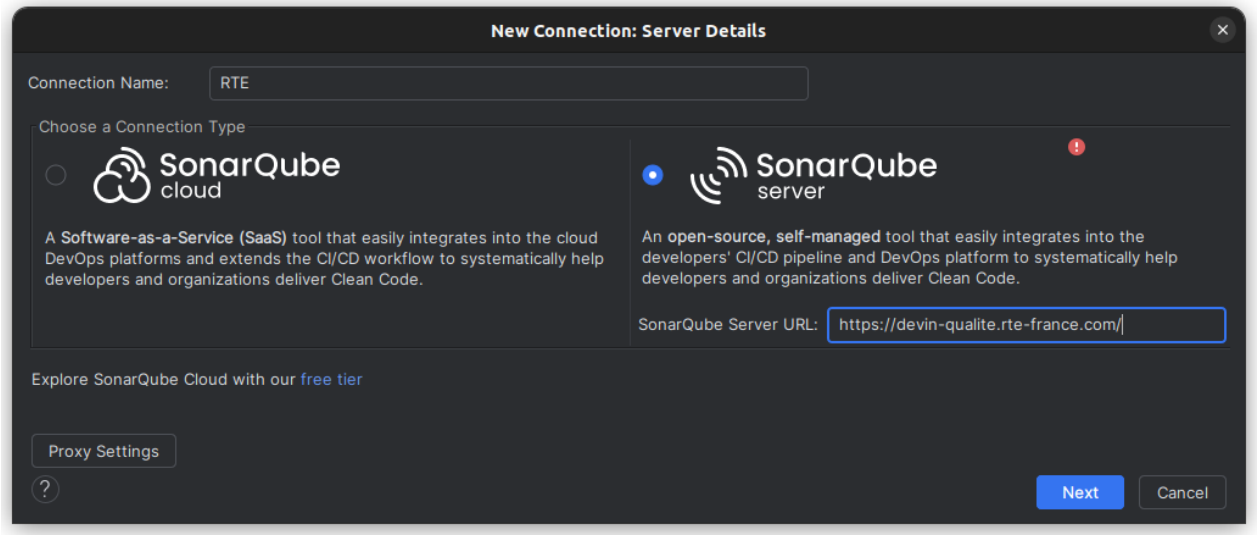
1. Une fois l'extension installée, ouvrez les paramètres de SonarQube for IDE
2. Dans l'onglet "Settings", vous verrez les options de configuration suivantes :
3. "Focus on new code" : permet de se concentrer sur l'analyse du nouveau code
4. "Automatically trigger analysis" : active l'analyse automatique en temps réel
5. Vérifiez que ces options sont configurées selon vos besoins



Étape 2 : Ajout d'une connexion au serveur SonarQube de RTE

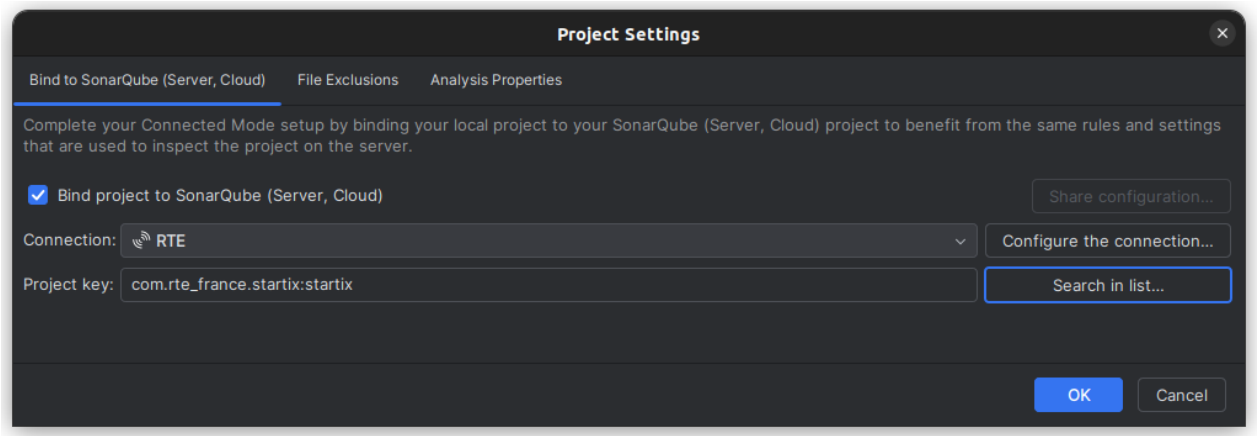
1. Dans la section "Connections to SonarQube", cliquez sur le bouton "+" pour ajouter une nouvelle connexion
2. Dans la fenêtre "New Connection: Server Details" :
3. Entrez un nom de connexion (par exemple "RTE") dans le champ "Connection Name"
4. Sélectionnez "SonarQube server" comme type de connexion
5. Entrez l'URL du serveur SonarQube de RTE : `https://devin-qualite.rte-france.com/` dans le champ "SonarQube Server URL"

6. Cliquez sur "Next" pour continuer



Étape 3 : Configuration du projet

1. Dans la fenêtre "Project Settings", accédez à l'onglet "Bind to SonarQube (Server, Cloud)"
2. Cochez la case "Bind project to SonarQube (Server, Cloud)"
3. Sélectionnez la connexion "RTE" dans le menu déroulant "Connection"
4. Entrez la clé du projet SonarQube dans le champ "Project key" (par exemple "com.rte_france.startix.startix")
5. Cliquez sur "Search in list..." pour vérifier que le projet existe sur le serveur
6. Cliquez sur "OK" pour finaliser la configuration



Utilisation de SonarQube pour IDE

Une fois configuré, SonarQube pour IDE analysera automatiquement votre code à mesure que vous l'écrivez. Les problèmes détectés seront affichés directement dans votre éditeur de code, avec des suggestions de correction.

Fonctionnalités principales

- Analyse en temps réel du code pour détecter les problèmes de qualité et de sécurité
- Explication détaillée des problèmes détectés
- Suggestions de correction (QuickFix)
- Partage des règles et paramètres avec votre équipe via la connexion au serveur SonarQube

Documentation Vault

À propos

Vault est un outil de gestion des secrets conçu pour sécuriser, stocker et contrôler l'accès aux informations sensibles.

Contexte

Actuellement dans notre chaine les informations sensibles qui sont nécessaires pour le bon déploiement et le bon fonctionnement d'une application, sont stockées dans un fichier private.conf sur chacune des machines projets, et ce fichier est alimenté manuellement par les projets.

L'utilisation de Vault va nous permettre de centraliser ces informations sensibles, et de mettre en place une récupération automatique de ces informations via Jenkins.

Selon le type de déploiement du projet, le processus sera le suivant :

1. Deploiement Ansible

Avant la phase de déploiement, Jenkins va récupérer les informations sensibles stockées sur Vault et permettre leur utilisation durant le déploiement. Au début de la phase de déploiement, on génère un fichier private.conf qui sera utilisé de la même manière que celui qui est utilisé actuellement. (Voir la section dédiée pour plus de détails).

2. Deploiement Kubernetes (K8S)

Avant la phase de déploiement, Jenkins va récupérer les informations sensibles stockées sur Vault. Puis on génère un fichier private.conf qui sera utilisé pour créer et deployer la ressource `secret` en utilisant `kubectl`. (Voir la section dédiée pour plus de détails).

Tutoriel / Intégration

1. Demande d'espace projet Vault

Voir la section dédiée.

2. Création des secrets dans l'espace projets Vault

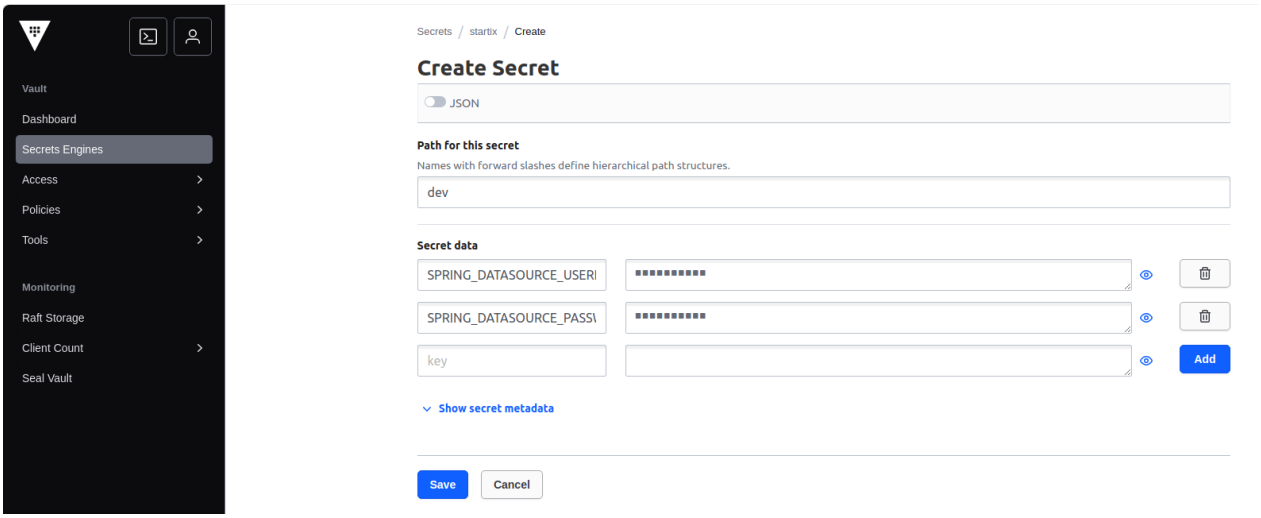
2.1. A partir de l'IHM

Une fois authentifié à vault, vous verrez apparaître votre espace projet dans l'onglet "**Secrets Engines**".

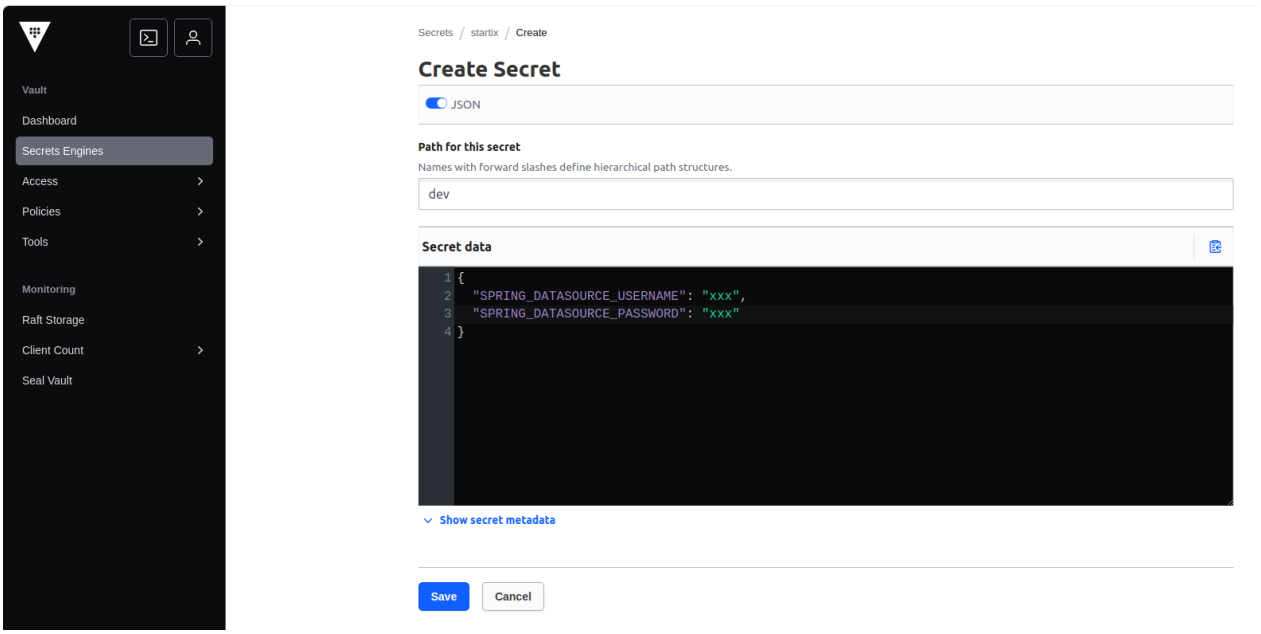
Dans votre espace projet vous avez la possibilité de créer des secrets en cliquant sur "**Create secret**". Les secrets que vous définissez peuvent contenir une ensemble de clé/valeur qui correspondent à vos informations sensibles.

Exemple : Création d'un secret

Dans cet exemple on fait le choix de créer un secret avec le path **dev** qui va contenir toutes les informations sensibles (sous forme de clé/valeur) dont on a besoin pour déployer notre application sur l'environnement de dev.



2.2. A partir d'un json



- Assurez-vous que "**JSON" est cochée**.
- Coller le contenu de votre json dans "Version data".

2.3. Utilisation du secretLoader

NOTE : Le script secretLoader va pousser sur Vault uniquement les secrets présent sur la machine sur laquelle vous avez exécuté le script.

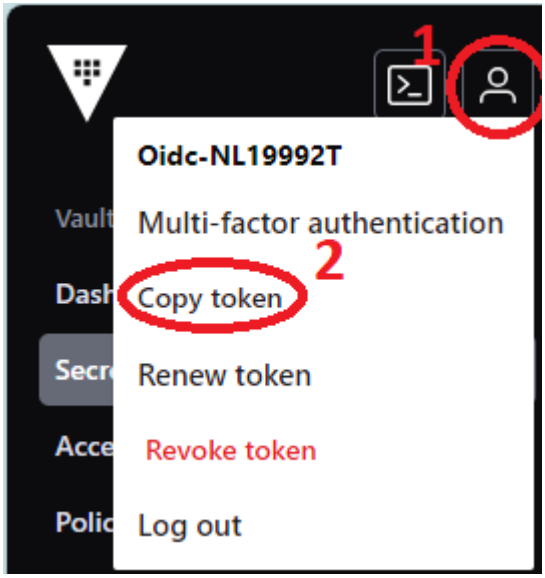
- Sur la machine où est stocké le fichier de configuration de variable :

```
## en root
curl -o secretLoader.sh https://gopro-collaboratif.rte-france.com/download/attachments/418425961/secretLoader.
```

- Puis exécuter le script et suivez les instructions :

```
## en root
chmod u+x secretLoader.sh
./secretLoader.sh
```

Durant l'exécution du script, trois instructions vont vous être demandées : - Chemin vers le fichier de configuration à uploader (ex: /applis/livraisons/private.conf) - Chemin du secret dans Vault (ex : cetautomatix/recette) - Token d'accès à Vault : vous pouvez récupérer votre token via l'IHM après vous être connecté à Vault en OIDC



Après avoir renseigné les instructions, le script doit vous retourner : "**Fichier JSON envoyé avec succès à Vault.**"

3. Configuration de la pipeline Jenkins

Pour utiliser Vault, il est nécessaire de modifier la pipeline de déploiement de l'application.

Les valeurs des secrets sont récupérées dans la variable secretValuesMap

Vous pouvez récupérer la valeur d'un secret en l'appelant de cette manière par exemple : {{secretValuesMap.SPRING_DATASOURCE_USERNAME}} (où SPRING_DATASOURCE_USERNAME est le nom de la clé dans vault)

3.1. Définition des constantes

Dans vos **Jenkinsfile_ICDC** et **Jenkinsfile_Deploy** il va falloir définir les constantes suivantes : (Voir le fichier Jenkinsfile_Deploy)

```
/** L'identifiant Jenkins du credential contenant les identifiants de connexion à vault **/
static final String VAULT_CREDENTIAL_ID = "$NOM_PROJET-vault-approle"

/** L'identifiant Jenkins du credential contenant le fichier de secret ansible vault **/
static final String ANSIBLE_VAULT_SECRET_FILE_ID = "ansible-vault-$NOM_PROJET"

/** Map des secrets nécessaires au fcontionnement de l'application **/
static final List<String> SECRET_LIST = [environnementCible]

// TODO: remplacer les valeurs d'environnement par les environnements sur lesquels vous utilisez vault
boolean useVault = (environnementCible in ['dev', 'recette'])
```

3.2. Configuration de la fonction de déploiement

3.2.1. Déploiement Ansible

- **Configuration de la fonction de déploiement**

Dans votre **Jenkinsfile_ICDC** et **Jenkinsfile_Deploy**, il va falloir modifier la configuration de la fonction deploy() :

```
stageDevin ('🏗️ Deploiement', doitDeployer) {
    utils.appendBuildDescription("Deploiement de la version <b>${versionDev}</b> à partir de <b>${currentBranc

    // Lancement du deploiement via l'infrastructure mutualisée.
    deploy {
            ansibleVersion = '2.11'
            targetEnv = environnementCible
            version = versionDev
            deployCredentialsId = SSH_KEY_CREDENTIALS_ID
            if ( useVault ) {
                secretList =  SECRET_LIST //ajout de la Liste des secrets dans Vault
                vaultCredentialId = VAULT_CREDENTIAL_ID //ajout du credential id vault
                projectName = NOM_PROJET //ajout du nom de projet utilisé pour récupérer les secrets dans l'es
                ansibleVaultSecretFileID = ANSIBLE_VAULT_SECRET_FILE_ID //utilisé pour chiffer les valeurs des
            }
    }
}
```

- **Configuration dans le code ansible**

Récupération du rôle createPrivateConf

Il est nécessaire de créer un rôle permettant la création du fichier private.conf. Vous pouvez intégrer ce rôle en copiant celui présent ici : roles/createPrivateConf

Ce rôle permet la création du fichier private.conf à partir d'un template. Il est nécessaire d'adapter ce template au besoin de votre projet, en remplaçant le nom des clés par celles qui sont contenues par votre secret dans Vault.

Appel du rôle dans votre script de déploiement

Dans votre script de déploiement, généralement site.yml il va falloir appeler le rôle que vous venez de créer :

```
- hosts: cetautomatix
  roles:
    - role: common/createPrivateConf
    vars:
        template_path: "{{ playbook_dir }}/templates/private.conf.j2" # Chemin vers votre template jinja2 priv
        secrets_path: "/applis/livraisons/private.conf"  # Facultatif, chemin par défaut : /applis/livraisons/
    when: secretValuesMap is defined
    - common/sanity-check
```

Pour le bon assurer le bon déroulement de votre déploiement, il est important que ce rôle soit la première tâche à être exécutée, car les étapes qui le suivent peuvent nécessiter de lire des informations dans le private.conf

Si vous utilisez la librairie **ansible-galaxy devin-common**, vous pouvez utiliser le rôle createPrivateConf de la manière suivante :

```
// Utilisation avec la librairie ansible-galaxy devin-common
- hosts: cetautomatix
  roles:
    - role: devin.common.createPrivateConf
      vars:
        template_path: "{{ playbook_dir }}/templates/private.conf.j2" # Chemin vers votre template jinja2 priv
        secrets_path: "/applis/livraisons/private.conf"  # Facultatif, chemin par défaut : /applis/livraisons/
      when: secretValuesMap is defined
    - devin.common.sanityCheck
```

3.2.2. Déploiement Kubernetes

- **Récupération des secrets depuis Vault**

Avant la phase de déploiement k8s, ajouter le phase qui permet de récuperer les secrets depuis l'espace Vault du projet de l'environnement cible ensuite génére un fichier private.env qui contient les secrets disponibles.

```
stageDevin("🏗️ Récupération des secrets", useVault) {
    echo "🚀 Récupération des secrets ${NOM_PROJET} sur ${SECRET_LIST}."

    def secrets = vaultUtils.getSecretsValues(
            NOM_PROJET,
            SECRET_LIST,
            VAULT_CREDENTIAL_ID
    )

    // Transformer la Map en fichier KEY=VALUE
    def envContent = secrets.collect { k, v -> "${k}=${v}" }.join("\n")
    writeFile file: 'private.env', text: envContent

    // Echo supplémentaire pour debug
    echo "📄 Fichier private.env généré avec ${secrets.size()} entrées"
}
```

- **Création du Secret sur K8S**

La phase suivante permet de créer une ressource de type `secret` dans kubernetes.

```
env.SECRETNAME = "${NOM_PROJET}-secret"

stageDevin("🏗️ Création du Secret Kubernetes", useVault) {
    echo "🚀 Création du secret ${SECRETNAME}."
    withKubeConfig([credentialsId: "jenkins-ic-startix-${NAMESPACE}"]) {
        sh '''
        kubectl create secret generic ${SECRETNAME} \
          --from-env-file=private.env \
          --dry-run=client -o yaml | kubectl apply -f -
        '''
    }
}
```

NOTE : Le $SECRETNAME doit correspondre à la secretRef dans deployment.yaml de votre configuration kubernetes.

Administratif

Dépendances

Référence	Intitulé de l'offre de service	Descriptif court	Pages documentaires associées
D1	Vault	Plateformes OPF	- https://gopro-tickets.rte-france.com/browse/DEVMCO-8209 - https://gopro-collaboratif.rte-france.com/display/DEV/Vault

D1. Demande d'accès et de création d'espace projet Vault.

Pour utiliser Vault, il faut au préalable que l'équipe DevOps définisse votre espace projet. Pour cela, vous pouvez sollicitez l'équipe DevOps soit par mail soit via GoPro, en respectant les règles de sollicitations suivantes :

- GoPro : Règles de sollicitation de l'équipe Dev!n via GOPRO

• BAL DevOps : Règles de sollicitation de la BAL

Voici un exemple de la demande Vault pour le projet Startix :

DEVIN MCO / DEVMCO-8209

Demande d'accès et de création d'un espace projet sur Vault

Modifier | Ajouter un commentaire | Rechercher sur un tableau | Plus | En validation

Informations

Type:	Demande de Support	Résolution:	Non résolu
Priorité:	Majeur	Version(s) corrigée(s):	Aucune
Affecte la/les version(s):	Aucune		
Composants:	Vault		
Étiquettes:	Aucune		
Nature:	Habilitation		
Nom Projet:	STARTIX		
Environnement:	OPF		

Description

Bonjour,

L'équipe **STARTIX** souhaite utiliser Vault pour la gestion des secrets sur l'environnement OPF.

Demande :

- Création d'un espace STARTIX dans Vault.
- Ajout des droits d'accès nécessaires à mon compte utilisateur.

On souhaite par la suite intégrer Vault avec notre pipeline de déploiement Jenkins.

Liens :

- Git Startix : https://devin-source.rte-france.com/startix/startix
- Jenkins Startix : https://devin-ic.rte-france.com/job/startix/job/startix

Une fois l'espace projet sera créé, vous pourrez accéder à Vault à l'url suivante : https://devin-vault.rte-france.com/
Pour vous authentifier, il suffit de sélectionner la méthode "**OIDC**" dans la liste déroulante et de cliquer sur "**Sign in with OIDC provider**". Vous serez rediriger vers la page d'authentification GAIA dans laquelle vous pourrez renseigner vos identifiants.

```
Seules les espaces projets des applications pour lesquelles vous serez habilités seront visibles.

Les habilitations d'accès à un espace vault sont attribuées automatiquement aux utilisateurs en fonction de le

- Utilisateur owner du projet sur gitlab : droits de lecture et écriture sur l'espace projet vault.
- Utilisateur maintener du projet sur gitlab : droits de lecture et écriture sur l'espace projet vault.
- Utilisateur developer du projet sur gitlab : droits de lecture sur l'espace projet vault.

Les utilisateurs n'ayant pas une de ces trois permissions sur le projet gitlab, n'auront pas d'autorisation d'
```